Luis Eduardo Mendoza Morales

Una Contribución a las Técnicas Avanzadas de Verificación de Procesos de Negocio y Sistemas Software Abiertos

– Tesis Doctoral –

Dirigida por:
Manuel I. Capel Tuñón

Marzo 2011

Universidad de Granada
Departamento de Lenguajes y Sistemas Infomáticos

Editor: Editorial de la Universidad de Granada
Autor: Luis Eduardo Mendoza Morales
D.L.: GR 3145-2011
ISBN: 978-84-694-4457-3

UNIVERSIDAD DE GRANADA
DEPARTAMENTO DE
LENGUAJES Y SISTEMAS INFORMÁTICOS

Manuel I. Capel Tuñón, Catedrático de Universidad, con N.R.P. 2595731568A0500, con destino actualmente en el Departamento de Lenguajes y Sistemas Informáticos y profesor–tutor del Programa de Doctorado P39/56/1 DESARROLLO DE SISTEMAS DE SOFTWARE (2010/2011) del Departamento de Lenguajes y Sistemas Informáticos.

INFORMA:

He leido y recomiendo al Departamento la aceptación de la Tesis Doctoral titulada "**Una Contribución a las Técnicas Avanzadas de Verificación de Procesos de Negocio y Sistemas Software Abiertos**", realizada bajo mi dirección por **Luis Eduardo Mendoza Morales**, como cumplimiento de los requisitos para acceder a la lectura que establece la Universidad de Granada para obtener la Titulación de Doctor.

Fechado: Marzo 2011

PhD. Manuel I. Capel Tuñón

Resumen

La búsqueda de notaciones y lenguales para la descripción de Procesos de Negocio (*Business Processes*, BP) ha sido uno de los objetivos del Modelado de Procesos de Negocio (*Business Process Modelling*, BPM) en los últimos tiempos. Como resultado de este esfuerzo, la Notación para el Modelado de Procesos de Negocio (*Business Process Modelling Notation*, BPMN) ha surgido como una notación gráfica estandarizada ampliamente aceptada para la documentación de BP. Sin embargo, por la heterogeneidad de sus construcciones y por la falta de una definición precisa de la notación, la especificación, el modelado y la verificación formal de BP se ve obstaculizado.

En esta tesis doctoral, como parte de los trabajos relacionados con las *Herramientas Metodológicas para la Verificación de Sistemas Abiertos y Procesos de Negocio*, se presenta una propuesta para la especificación, el modelado y la verificación composicional de requisitos no funcionales y restricciones temporales de modelos de BP realizados con BPMN. Para la especificación y el modelado se define una semántica formal de los elementos notacionales de tiempo de BPMN, bajo el dominio semántico del cálculo de proceso basado en CSP, *Communicating Sequential Processes + Time*, CSP+T. Para la verificación formal de los modelos de BP se utiliza el *Enfoque Formal de Verificación Composicional* (EFVC), una propuesta que se hace en el ámbito de la Ingeniería de Software (IS) para la verificación composicional de Sistemas con Criticidad en Seguridad (*Safety–Critical Systems*, SCS), que integra la verificación composicional con la técnica de verificación automática *Model Ckecking* (MC). Como resultado, se ha obtenido una herramienta metodológica que incorpora avances de la IS al ámbito del BPM para el análisis formal de modelos de BP.

Además, se presenta el diseño general del conjunto de herramientas de software que permiten automatizar el EFVC y se describe la implementación de la herramienta BTransformer, la cual soporta la transformación de modelos BPMN a términos de proceso CSP+T. Adicionalmente, se discute la aplicación de los aportes de la presente tesis doctoral a dos casos de interés empresarial e industrial. Uno, relacionado con un BP considerado crítico para la estrategia de Gestión de las Relaciones con el Cliente (*Customer Relationship Management*, CRM). El otro, corresponde al sistema que implementa el proceso crítico del protocolo de comunicación que constituye el corazón de una red de comunicación de teléfonos móviles. La aplicación del

EFVC nos ha permitido corroborar que éste puede ser utilizado para verificar tanto el comportamiento de BP con criticidad, como los componentes que conforman el software que implementa BP complejos y con criticidad, conformando así una contribución dentro del área de las *Técnicas Avanzadas de Verificación de BP y Sistemas Software Abiertos*.

Granada, Marzo 2011 *Luis E. Mendoza Morales*

Tabla de Contenidos

CONTEXTO DEL TRABAJO

ASPECTOS TEÓRICOS Y CONCEPTUALES

Lista de Figuras

Lista de Tablas

Abreviaciones

ebXML	e–business XML.
ACP	Algebra of Communicating Processes.
ADF	Activity-Decision Flor.
AMBER	Architectural Modelling Box for Enterprise Redesign.
AMMA	ATLAS Model Management Architecture.
AP	Atomic Propositions.
AT	Autómata Temporizado.
ATL	ATLAS Transformation Language.
B2B	Business to Business.
BDD	Ordered Binary Decision Diagram.
BMC	Bounded Model Checking.
BP	Business Processes.
BPD	Business Process Diagram.
BPM	Business Process Modelling.
BPMI	Business Process Management Initiative.
BPML	Business Process Modelling Language.
BPMN	Business Process Modelling Notation.
BPMS	Business Process Management Systems.
BPQL	Business Process Query Language.
BPSS	Business Process Specification Schema.
BPTM	BP Task Model.
BTS	Base Transceiver Stations.
CCS	Calculus of Communicating Systems.
CFM	Common Formal Model.
CPN	Coloured Petri Nets.
CRM	Customer Relationship Management.
CSP	Communicating Sequential Processes.
CSP+T	Communicating Sequential Processes + Time.
CTL*	Computational Tree Logic.
CCTL	Clocked Computation Tree Logic.
DB	Data Base.
DDBM	Distributed Data Base Manager.
DFD	Data Flow Diagram.
DFS	Depth–First–Search.
EDOC	Enterprise Distributed Object Computing Business Processes.
EFVC	Enfoque Formal de Verificación Composicional.
EI	Estructura de Intervalo.
EIS	Enterprise Information Systems.

EK	Estructura de Kripke.
EPC	Event–Process Chains.
ER	Entity–Relationship.
ERP	Enterprise Resource Planning.
FDR2	Failures–Divergence Refinement versión 2.
FTS	Fair Transition Systems.
GPN	Gestión de Procesos de Negocio.
HyTech	HYbrid TECHnology.
ICAM	Integrated Computer–Aided Manufacturing.
ICOM	Input–Control–Output–Mechanism.
IDE	Integrated Development Environment.
IDEF	Integrated Definition for Function Modelling.
IS	Ingeniería del Software.
JDK	Java Development Kit.
LISI	Laboratorio de Investigación en Sistemas de Información.
LTL	Linear Temporal Logic.
LTS	Labelling Transition System.
MEDISTAM–RT	Método para el Diseño de Sistemas basado en Transformación Analítica de Modelos de Tiempo Real.
MC	Model Ckecking.
MLTL	Metric and Layered Temporal Logic.
MOF	Meta Object Facility.
MTL	Model Transformation Language.
OCL	Object Constraint Language.
OMG	Object Management Group.
OO	Object–Oriented.
ParaDiSe	Parallel & Distributed Systems Laboratory.
PLC	Programmable Logic Controllers.
PN	Petri Nets.
POR	Partial Order Reduction.
PROMELA	Process or Protocol Meta Language.
QoS	Quality of Service.
QVT	Query/View/Transformation.
RAD	Role Activity Diagram.
RAG	Razonamiento Asumir/Garantizar.
RF	Requisito Funcional.
RFP	Request For Proposal.
RID	Role Interaction Diagram.
RNF	Requisito No Funcional.
SAT	Boolean SATisfiability Problem.
SCS	Safety–Critical System.
SDK	Software Development Kit.
SE	Software Engineer.
SLA	Service Level Agreement.
SMC	Symmetry–based Model Checker.

SMV	Symbolic Model Verifier.
SOA	Service Oriented Architecture.
SOS	Structural Operational Semantics.
SPIN	Simple Promela Interpreter.
ST	State Transition.
STR	Sistemas de Tiempo Real.
STRIM	Systematic Technique for Role & Interaction Modelling.
TCTL	Timed Computation Tree Logic.
TGS	Teoría General de los Sistemas.
TIC	Tecnologías de la Información y la Comunicación.
TP	Theorem Proving.
UDDI	Universal Description, Discovery and Integration.
UI	User Interface.
UML	Unified Modelling Language.
UML–RT	UML for Real Time.
UML–TSM	UML Timed State Machine.
UPPAAL	Universidad de UPPsala + Universidad de AALborg.
VERBUS	VERification for BUSiness processes.
VPS	Vender Producto/Servicio.
WF	Workflow.
WfMC	Workflow Management Coalition.
WfMS	Sistemas de Gestión de Workflow.
WS	Web Services.
WS–BPEL	WS Business Process Execution Language.
XML	Extensible Markup Language.
XPDL	XML Process Definition Language.

CONTEXTO DEL TRABAJO

Capítulo 1
Introducción

Resumen El objeto de este capítulo es contextualizar el desarrollo de esta tesis doctoral. Para definir con precisión su alcance, en la sección 1.1 describimos nuestra motivación y los antecedentes, esbozando el problema a resolver. Luego, en la sección 1.2, discutimos los trabajos relacionados y determinamos algunas debilidades de éstos, a fin de enmarcar nuestros objetivos. A continuación, en la sección 1.3, enumeramos los objetivos planteados en el desarrollo del trabajo, seguido del punto 1.4, donde concretamos las aportaciones de la tesis a la comunidad científica. Finalmente, en la sección 1.5, se hace la reseña de la estructura del libro.

1.1 Motivación y antecedentes

Una pregunta difícil de contestar para las organizaciones es cómo crear Procesos de Negocio (*Business Processes*, BP) robustos y flexibles. Un BP es definido normalmente como el conjunto de actividades inter relacionadas con el objetivo de agregarle valor al cliente [67]. La *Gestión de Procesos de Negocio* (GPN) se ha venido perfilando como un área que comprende las actividades que pueden realizar las organizaciones para optimizar sus BP o para adaptarlos a las nuevas necesidades de los clientes. En concreto, la GPN define las actividades a realizar por las organizaciones para gestionar y mejorar sus BP. Las actividades de la GPN abarcan el análisis, el diseño y la definición de BP. En este contexto, la definición de notaciones y lenguajes para modelar BP es un aspecto clave dentro de la GPN, debido a la necesidad de describir la estructura y el comportamiento esperado de los BP.

Por otro lado, la brecha entre los negocios y la *Ingeniería de Software* (IS) ha sido uno de los mayores problemas en la industria del software [67, 120]. La dificultad de comunicación entre los empresarios y las personas que desarrollan software es la causa principal de la no satisfacción de los requisitos empresariales por parte del software implementado [67, 120]. Por ello, la disciplina de Modelado de Procesos de Negocio (*Business Process Modelling*, BPM) ha sido el medio que ha permitido establecer los puentes entre las empresas y la IS [90]. Entre otros, el BPM ha tenido por objetivo lograr la descripción de BP con una notación o lenguaje que no sea sólo compren-

sible por los empresarios, sino también sea lo suficientemente riguroso para poder derivar el software que soporta la ejecución de los BP. Un lenguaje que facilite la definición de los requisitos del software que soporte el negocio, que sea comprendido por los empresarios y por los ingenieros de software, permite compartir el conocimiento del dominio de los negocios, los requerimientos de negocio y llevar a cabo su verificación [90].

La Notación para el Modelado de Procesos de Negocio (*Business Process Modelling Notation*, BPMN) [158], inicialmente desarrollada por la organización Iniciativa de Gestión de Procesos de Negocio (*Business Process Management Initiative*[1], BPMI) y actualmente mantenida por *Object Management Group*[2] (OMG), ha venido a ser una notación gráfica estandarizada que permite describir el BPM en un formato basado en flujos de trabajo [158] (en inglés, *workflow*). La versión actual de BPMN es la 1.2 y ya está disponible la versión Beta de la futura versión 2.0. BPMN es la notación más ampliamente aceptada para la documentación de BP en todo el mundo. BPMN se ha visto como una herramienta de comunicación por parte de todos los involucrados e interesados del negocio (englobados bajo el término *stakeholders*). Facilita la comunicación entre los analistas de negocio (quienes definen y diseñan los procesos), los desarrolladores de software (responsables de implementar los procesos) y los gerentes y administradores del negocio (quienes monitorean y gestionan los procesos), manteniendo al mismo tiempo el más alto nivel de integridad de los procesos [158].

Sin embargo, el análisis semántico de los modelos BPMN se ve dificultado por la falta de una definición precisa de la notación y por la diversidad de construcciones que pueden realizarse. Si bien las reglas sintácticas están ampliamente documentados en la especificación estándar BPMN [158], la semántica es descrita en forma narrativa utilizando una terminología que puede resultar ambigua desde el punto de vista matemático–formal. En particular, estamos interesados en la verificación formal del comportamiento de los BP y en la incorporación de las restricciones temporales implícitas en éstos, tanto en su especificación como en su modelado.

En el contexto empresarial, el énfasis de los Acuerdos de Nivel de Servicio (*Service Level Agreement*, SLA) es sobre el tiempo de respuesta de los BP que garanticen un nivel óptimo de Calidad de Servicio (*Quality of Service*, QoS). No sólo se requiere que las actividades y tareas sean realizadas correctamente y en el orden correcto; sino que, además, la coordinación de tareas concurrentes de los BP no lleve a situaciones de bloqueo o error, y éstos sean completadas en los tiempos establecidos en los SLA. Los BP relacionados con la prestación de servicios típicamente deben diseñarse y

[1] http://www.bpmi.org/.

[2] http://www.omg.org/.

verificarse tomando en cuenta tiempos de respuesta iniciales y cumplimiento de tiempos máximos. Muchas veces la QoS de los BP es medida en función del tiempo máximo en el cual deben ejecutarse las actividades y del tiempo de duración esperado para los BP como un todo.

Adicionalmente, el aumento de la complejidad de los BP incrementa la probabilidad de que estos fallen. Estrategias como la introducción de mecanismos de gestión de fallos y compensación o la delegación de procesos a terceros vía remota, para tratar de disminuir los posibles fallos de los BP, se ha convertido en un aumento de la complejidad de éstos. Por ello, se hace necesario que se introduzcan herramientas automatizadas que sistematicen la corrección de las definiciones de los BP, antes de su implementación en sistemas que soporten su ejecución, tal como lo hacen los Sistemas de Gestión de Procesos de Negocio (*Business Process Management Systems*, BPMS) o los Sistemas de Información Empresariales (*Enterprise Information Systems*, EIS), entre otros.

En contraste con lo anterior, la IS ha venido consolidando, desde sus inicios, el uso de técnicas para detectar errores en programas de software, circuitos electrónicos del hardware y protocolos de comunicación, entre otros. En este sentido, nosotros creemos que la verificación automática (*Model Checking*, MC) también puede ser aplicada en la disciplina de BPM, con el objeto de detectar errores en las etapas de análisis, diseño y definición de BP; es decir, antes de su implantación a través de sistemas de uso empresarial o industrial. Con el MC podemos demostrar de forma automatizada la presencia o ausencia de determinados tipos de errores derivados del cumplimiento de propiedades esperadas para los sistemas o procesos con criticidad. En particular, aquellos relacionados con requisitos no funcionales y de rendimiento donde se establezcan restricciones temporales.

La verificación de BP es un tema que cada día abre más posibilidades de investigación debido a las distintas propuestas de lenguajes y notaciones que han surgido en los últimos tiempos para tratar de estandarizar su definición [151, 220]. La revisión de antecedentes revela que se han realizado pocos trabajos enfocados a la verificación de requisitos temporales de BP. En síntesis, podemos observar tres grandes etapas.

La primera, cuyos orígenes se remontan a la definición, modelado y verificación de métodos basados en el análisis de flujos de trabajo. Específicamente al tratamiento que se les puede dar a estos dentro de los Sistemas de Gestión de Workflow (*Workflow Management Systems*, WfMS) y los BPMS. Estos sistemas están constituidos principalmente por un motor de workflow; un mecanismo que ejecuta la definición previa de los BP en un lenguaje de programación adecuado para ello. Con la finalidad de garantizar que el producto desarrollado cumpliese lo más fielmente los BP, los investigadores en IS han buscado definir patrones de comportamiento repetibles y

precisar la semántica de los workflow de tal manera que se facilite su verificación e implementación.

La segunda, representada por el auge del Lenguaje de Modelado Unificado (*Unified Modelling Language*, UML) y su posicionamiento como un estandar de facto en la IS. La mayoría de los estudios son acerca de la semántica y verificación de diagramas de actividad, dado que este artefacto de modelado aglutina los elementos fundamentales propuestos por UML para el modelado de workflow. Además, se considera que este diagrama es de fácil interpretación por parte de todos los stakeholders en la automatización de BP: usuarios, clientes y desarrolladores, entre otros.

La tercera, ya en estos últimos años, caracterizada por la necesidad de los negocios de prestar mejores servicios sobre la plataforma de Internet. En esta etapa, la IS ha aprovechado los avances en Arquitecturas Orientadas a Servicio (*Service Oriented Architecture*, SOA) y surgieron los Servicios Web (*Web Services*, WS). Esto conllevó a la formulación del Lenguaje de Ejecución de Procesos de Negocio con Servicios Web (*WS Business Process Execution Language*, WS–BPEL), el cual se ha convertido en el estándar para la definición del concepto de orquestación de WS. Finalmente, como resultado del impacto que tuvo WS–BPEL, al poco tiempo surgió BPMN. La consolidación de WS–BPEL y BPMN han abierto la posibilidad de analizar y verificar el comportamiento de BP.

Como comentario final, el problema de la verificación de requisitos temporales de BP está muy relacionado con la verificación de requisitos temporales de coordinaciones y composiciones de WS, ya que éstas son dependientes de aspectos de sincronización, donde las restricciones temporales juegan un papel muy importante. Soluciones que se propongan a nivel de colaboración y coordinación de BP pueden ser extrapolables a la orquestación de WS.

1.2 Discusión de trabajos relacionados

La revisión y análisis de trabajos que se presenta a continuación se hace cronológicamente en base a las etapas descritas anteriormente y sobre aquellos trabajos con mayor relevancia en relación a *Técnicas para la Verificación de BP*, que guardan una íntima vinculación con la especificación, el modelado y la verificación de *Sistemas Software Abiertos*, y que permiten precisar (ver secciones 1.3 y 1.4) el aporte de esta tesis doctoral.

1.2.1 Analizador y redes de flujo de trabajo

El analizador de flujo de trabajo (Workflow analyzer, Woflan) [206, 207] es una herramienta de verificación producto de un proyecto de investigación, cuyos inicios datan de finales de 1996. Analiza la definición de procesos de workflow generados por distintos productos comerciales, usando técnicas de análisis basados en redes de Petri. Utiliza una subclase de redes de Petri, las Redes de flujo de trabajo (*Workflow Nets*, WF–nets) [205], usadas para modelar las definiciones de procesos workflow propuestos originalmente por W.M.P. van der Aalst [1].

El principal objetivo de Woflan es verificar la correctitud de un workflow especificado en algún WfMS, traduciendo directamente estas especificaciones a redes de Petri. Los autores justifican el uso de las redes de Petri para acometer el problema de verificar workflow complejos por ser un lenguaje de modelado universal con fundamentos matemáticos sólidos [205, 206, 207]. Ellos indican que las redes de Petri son las adecuadas para la especificación del control del flujo.

Como consecuencia de la decisión de centrarse en la perspectiva del control del flujo dentro de los workflow, las investigaciones que han contribuido a esta herramienta sólo trabajan sobre aspectos relacionados con la secuencia de tareas dentro de un workflow. Por ello, la herramienta puede generar secuencias de errores comportamentales, proporcionando la secuencia mínima de tareas cuya ejecución llevan inevitablemete a un error. Aunque consideran que la duración de actividades y las características temporales de 'disparadores' (del inglés, *triggers*) son cruciales cuando se analiza el rendimiento de los workflow, no contemplan la prueba de propiedades cuantitativas tales como tiempo de respuesta, tiempos de espera y tasas de ocupación.

La última versión que se conoce de Woflan, la 3.0, es descrita en la tesis doctoral de Verbeek [205], donde se presentan los resultados teóricos y prácticos usados para desarrollar la herramienta. Sin embargo, en su página web[3] sólo está disponible la versión 2.2.

1.2.2 Semántica y verificación de diagramas de actividad UML

El trabajo de Eshuis [68] muestra cómo el MC puede ser usado para verificar requisitos funcionales sobre una especificación de workflow realizada utilizando diagramas de actividad de UML. El objetivo es la definición de

[3] http://is.tm.tue.nl/research/woflan/

una semántica formal para los diagramas de actividad y que esa semántica permita la verificación de requisitos funcionales usando MC. En este sentido, sólo contemplan la verificación del control de flujo de los procesos de workflow.

Los autores indican que UML es muy útil por ser un lenguaje gráfico; de más fácil comprensión por la mayoría de la gente que las especificaciones textuales. Pero como los diagramas de actividad de UML carecen de una semántica formal, ellos definen una que es adecuada para el modelado de workflow. Resaltan la importancia de hacer la verificación de estos diagramas y lo insuficiente que pueden ser las pruebas a nivel de los sistemas que implementan los procesos. Desarrollan una herramienta que permite realizar verificaciones de diagramas de actividad UML incorparando los requisitos funcionales en la verificación, haciendo uso de la lógica temporal Lógica de Árbol Computacional (*Computational Tree Logic*, CTL*) para su especificación y de la herramienta Verificador de Modelo Simbólico (*Symbolic Model Verifier*, SMV) para su verificación. La herramienta desarrollada está orientada a gente no familiarizada con métodos formales.

Proponen dos semánticas formales diferentes. A nivel de requisitos, el diagrama de actividad es usado como una especificación de workflow. A nivel de implementación, la especificación del workflow establece cómo el sistema de workflow debe comportarse. Aunque toman como semánticas de partida los Diagramas de Estados (en inglés, *statecharts*) y las redes de Petri, después de un exhaustivo análisis y tomando en cuenta que los BP en sí mismos son sistemas reactivos, basan las semánticas propuestas en los statecharts. Reconocen que las semánticas propuestas son más difíciles de analizar que las redes de Petri, argumentando que las semánticas basadas en redes de Petri no son adecuadas para sistemas reactivos [68].

1.2.3 Modelo para la verificación formal de procesos de negocio

En la tesis doctoral de Arias [69] se define un modelo para la verificación formal de BP; específicamente de requisitos funcionales de BP. Proponen una arquitectura abierta, modular y extensible para la verificación de BP y composiciones de WS, que permite la integración de distintos lenguajes de definición de procesos y herramientas de verificación. Se basa en la propuesta de un Modelo Formal Común (*Common Formal Model*, CFM), que se fundamenta en sistemas de estado-transición etiquetados, pero con una notación y abstracción orientadas a la representación de BP para las perspectivas de control de flujo y datos. En consecuencia, identifican un conjunto de requisitos para la definición de especificaciones de BP y composiciones

WS, así como los requisitos que cualquier modelo de BP debería cumplir, basados en el concepto de *solidez* (del inglés, *soundness*).

Con el objeto de comprobar la expresividad y adecuación del formalismo propuesto para la representación de los BP, realizan un análisis basado en los patrones de workflow de W.M.P. van der Aalst [6]. Adicionalmente, se integra WS–BPEL (conocido para la época como *Business Process Execution Language for Web Services* —BPEL4WS) en la arquitectura. Para ello, se define su semántica en términos del formalismo CFM, así como una metodología de transformación de definiciones de procesos BPEL4WS a definiciones CFM. El énfasis del trabajo es sobre los aspectos de coreografía de composiciones de WS, dejando al lado aspectos relativos a la colaboración entre BP. En consecuencia, no contemplan la verificación de concurrencia de BP que colaboran y se comunican, ni la verificación de requisitos de rendimiento relativos a restricciones temporales de BP. Sólo se centran en la verificación de las perspectivas de control de flujo y datos.

También se integran en la arquitectura dos herramientas de MC: Interpretador Promela Simple (*Simple Promela Interpreter*, SPIN) y NuSMV, una extensión de SMV. Para ello, se define una transformación entre definiciones CFM y los lenguajes de entrada de estas herramientas. Además, la arquitectura propuesta ha sido implementada en el prototipo Verificación para Procesos de Negocio (*VERification for BUSiness processes*, VERBUS) [70], el cual está disponible en su pagina web[4].

1.2.4 Enfoque algebraíco de procesos y formalización de BPMN

La formalización descrita en los trabajos de Wong y Gibbons [213, 215], según el autor, dan las bases para un modelo unificado para la especificación, análisis y verificación de procesos workflow, tanto a nivel de orquestación como de coreografía. Dan la formalización completa de los patrones de procesos workflow propuestos por W.M.P. van der Aalst [6] en el álgebra de procesos Procesos Secuenciales de Comunicación (*Communicating Sequential Processes*, CSP) [92, 184, 190]. Acometen tanto construcciones simples, como primitivas de enrutamiento complejas. Sin embargo, su enfoque está limitado al control de flujo estático. Modelan cada patrón de control de flujo en CSP.

En el trabajo [215] modelan la especificación e implementación de workflow como procesos CSP, facilitando el análisis composicional de procesos de workflow. Explotan las nociones de refinamiento de procesos de CSP para

[4] http://www.it.uc3m.es/jaf/verbus/

comparar workflow. Ya en este artículo se da una semántica para BPMN en CSP que permite a los diseñadores de procesos workflow construir especificaciones usando BPMN, y comparar formalmente diagramas BPMN.

Como una continuidad de los trabajos citados anteriormente, en [214, 216, 217] detallan un modelo semántico de BPMN en CSP y estudian una ampliación del modelo introduciendo información relativa del tiempo. El enfoque está especialmente diseñado para reflejar el transcurso del tiempo a lo largo de la ejecución de diagramas BPMN y ser verificados automáticamente con la herramienta de MC Refinamiento de Fallos–Divergencia (*Failures–Divergence Refinement*, FDR) [71]. Sin embargo, no establecen cómo especificar la verificación de propiedades de BP, tanto dependientes como independientes del tiempo. Sólo utilizan la información temporal de las tareas para precisar el orden de ejecución de éstas como parte del control de flujo de los BP.

Estos trabajos son los primeros que introducen formalmente una semántica de BP en CSP y describen cómo puede ser aplicada para razonar y refinar diagramas BPMN. Introducen un modelo de tiempo relativo [217] el cual extiende el modelo no temporizado [216] con una temporización relativa de ejecución y, usando esos dos modelos, discuten la noción de compatibilidad y la relación entre los dos modelos [214]. Explotan la noción de refinamiento de procesos de CSP para especificar y comparar sistemas workflow.

1.3 Objetivos del trabajo

Después de haber esbozado los antecedentes y el análisis de los trabajos relacionados, sobre la base de algunas debilidades encontradas en estos trabajos y bajo el contexto de la propuesta de *Herramientas Metodológicas para la Verificación de Sistemas Abiertos y BP*, los objetivos de la tesis doctoral fueron los siguientes:

1.3.1 General

Proponer una herramienta metodológica para la especificación, el modelado y la verificación composicional de *Procesos de Negocio*, basado en un enfoque que formaliza la descripción del comportamiento de las tareas, originalmente descritos en BPMN, universalmente aceptada en la comunidad.

Este objetivo resulta bastante general. Por ello, lo concretamos a través de las siguientes condiciones que debe cumplir la herramienta metodológica.

(a) *La definición de una semántica formal que represente sin ambigüedad el comportamiento de BP.*
La semántica formal debe especificar con *precisión* los elementos notacionales de BPMN que permita obtener una especificación fiel del comportamiento del BP.

(b) *La estructuración de un proceso de verificación composicional formal de requerimientos no funcionales de BP que incluya su comportamiento temporal.*
El proceso de verificación debe seguir las pautas de la verificación composicional formal y contemplar que la prueba de correctitud de BP con respecto a sus requisitos no funcionales y sus restricciones temporales se haga a partir de la prueba individual del comportamiento de los BP locales que ejecutan los participantes que colaboran para la realización de los BP.

(c) *La integración de un comprobador de modelos (en inglés, comprobador de modelos) al proceso de verificación composicional.*
El comprobador de modelos debe basarse en el lenguaje de especificación formal utilizado en la definición semántica de los elementos notacionales de BPMN y soportar el proceso de verificación composicional formal del comportamiento de BP.

1.3.2 Específicos

Para lograr con éxito el objetivo general, a continuación se enumeran los objetivos específicos que orientaron el desarrollo de la tesis.

1. *Analizar el estado del arte* relacionado con la especificación, el modelado y la verificación composicional formal, y su relación con los BP y el BPM.
2. *Proporcionar un análisis temporal de los elementos notacionales de BPMN* que sustente la ventaja del uso de las leyes de los cálculos de procesos para la definición de la semántica fornal de los elementos notacionales de BPMN.
3. *Definir la semántica de los elementos temporizados de BPMN* a través de un conjunto de reglas que permitan especificarlos con precisión, si-

guiendo la semántica de las leyes de los cálculos de procesos basadas en CSP.
4. *Conformar la base formal de la tecnología* (algoritmos, reglas, procedimientos, entre otros) que soporta la herramienta metodológica para la especificación, el modelado y la verificación composicional formal.
5. *Diseñar un proceso de verificación del comportamiento de BP* basado en un enfoque de verificación composicional formal.
6. *Desarrollar herramientas que permiten automatizar* (parte o todo) el proceso de verificación del comportamiento de BP.
7. *Integrar la técnica de MC*, y un comprobador de modelos que soporte cálculos de procesos basados en CSP, en el proceso de verificación composicional formal para probar la consistencia de los BP modelados con BPMN.
8. *Aplicar la herramienta metodológica* a situaciones de interés empresarial e industrial, con la finalidad de probar su viabilidad y su capacidad práctica.

1.4 Aportaciones de la tesis

El logro de los objetivos indicados anteriormente permite resumir la aportación de la tesis en los siguientes aspectos:

- ***Especificación, modelado y verificación composicional*** *del cumplimiento de requisitos de rendimiento (o no funcionales) de BP*. Para los negocios es tan importante la especificación y verificación de lo *qué debe hacerse* (requisitos funcionales) como *cómo de bien debe hacerse* (requisitos de rendimiento). Un aporte en ese sentido, es la utilización de una nueva notación basada en el cálculo de procesos CSP, conocida como Procesos Secuenciales de Comunicación + Tiempo (*Communicating Sequential Processes + Time*, CSP+T), para la verificación composicional de requisitos temporales del modelo de tareas de BP.
- ***Definición de una semántica formal*** *que precise los atributos temporales de todos los elementos involucrados en el control del flujo de los BP y en la colaboración entre BP, extensible a la composición de WS*. Hasta la fecha, aspectos como la duración de las tareas, tiempos de espera o de expiración, han sido abstraidos de la especificación, el modelado y la verificación de BP, en general, y de BP modelados con BPMN, en particular.

- ***Propuesta de un proceso formal de verificación composicional*** *que permita la prueba de requisitos no funcionales y restricciones temporales de BP*. Tradicionalmente, la incorporación del tiempo como objeto de verificación torna complejo el proceso de verificación de procesos. Se plantea entonces un nuevo enfoque de verificación composicional aplicable a los BP que aprovecha las fortalezas del cálculo de procesos CSP y la notación CSP+T, la cual está basada en CSP, y que plantea una alternativa de solución al problema de la explosión de estados (este problema será tratado posteriormente con mayor detalle).

En esta tesis doctoral pretendemos realizar aportaciones en las disciplinas de la especificación, el modelado y la verificación de propiedades no funcionales y temporales de la ejecución de BP, dentro del área de las Técnicas Avanzadas de Verificación de BP y Sistemas Software Abiertos. En primer lugar, definimos una semántica formal para el subconjunto de elementos notacionales de BPMN relacionados con aspectos temporales. La semántica se define como una nueva notación que relaciona los elementos BPMN temporizados y el lenguaje formal CSP+T. El uso de CSP+T se debe a que este lenguaje formal permite especificar el comportamiento de los elementos notacionales de BPMN, haciendo énfasis en sus aspectos temporales de una manera clara y precisa. Además, aprovechamos la capacidad del operador de composición de procesos de CSP para el modelado detallado y la verificación composicional de BP. En segundo lugar, describimos un *Enfoque Formal de Verificación Composicional* (EFVC) que permite llevar a cabo la comprobación de requisitos no funcionales y temporales de BP modelados con BPMN. Este enfoque integra, por un lado, el concepto de abstracción/refinamiento para la obtención de modelos detallados aplicando la semántica propuesta; y por el otro, el razonamiento composicional para la conducción de la verificación. De esta manera buscamos minimizar el problema de la explosión de estados por el impacto de la inclusión del tiempo como parte de la verificación de BP. En tercer lugar, proponemos la integración del comprobador de modelos FDR para soportar la verificación automática del comportamiento de los BP, como parte del proceso de aplicación del EFVC. En síntesis, la integración de las aportaciones descritas anteriormente resultan en la conformación del diseño de una Herramienta Metodológica que podemos aplicar tanto a la Verificación de Sistemas Abiertos como a BP.

1.5 Estructura del tomo

Además del presente capítulo de Introducción, el tomo está conformado por los siguientes capítulos:

Capítulo 2. Presenta el estado del arte que expone todos los elementos conceptuales que gitan en torno a la especificación, el modelado y la verificación de BP que utilizamos a lo largo de este trabajo, así como el uso de algunos métodos formales en el BPM y en la precisión formal de BP.

Capítulo 3. Describe la base formal que sustentan los elementos tecnológicos (algoritmos, reglas y procedimientos, entre otros) que soportan la herramienta metodológica propuesta, incluyendo el análisis temporal de los elementos notacionales de BPMN, como preámbulo a la defininión de la semántica que permite la especificación, el diseño y la verificación del comportamiento de BP modelados con BPMN.

Capítulo 4. Expone en detalle el EFVC que incluye nuestra propuesta y se muestra la validación del enfoque de verificación composicional. Se describe cómo el EFVC integra los distintos formalismos que permiten conducir la verificación, así como la visión que tenemos para su aplicación a los BP a través de la adaptación del EFVC para la verificación de BP.

Capítulo 5. Describe el diseño general que orienta la construcción del conjunto de herramientas que soportan la mayor parte del enfoque de verificación composicional que se propone en esta tesis. Además, se detallan las vistas más importantes de la arquitectura de la herramienta BTRANSFORMER, la cual implementa la semántica definida para precisar el comportamiento de BP modelados con BPMN.

Capítulo 6. Presenta la aplicación de nuestra propuesta a dos casos de interés empresarial e industrial. El primero, referido a un proceso crítico de la estrategia de negocio conocida como Gestión de las Relaciones con el Cliente (*Customer Relationship Management*, CRM); y el segundo, relacionado con el proceso crítico del protocolo de comunicación entre las distintas Estaciones Base de Transmisores (*Base Transceiver Stations*, BTS) que constituyen una red de comunicación de teléfonos móviles.

Capítulo 7. Muestra la discusión de los resultados obtenidos a lo largo del desarrollo de la tesis doctoral, se enumeran las conclusiones, y se indican algunos aspectos relacionados con el trabajo futuro.

ASPECTOS TEÓRICOS Y CONCEPTUALES

Capítulo 2
Estado del Arte

Resumen El objeto de este capítulo es presentar el resultado del estudio sobre el estado actual de los tópicos más importantes relacionadas con las contribuciones propuestas en esta tesis doctoral. En la sección 2.1 se describen las lógicas temporales más importantes conocidas a la fecha, resaltando la que utilizaremos como parte de nuestra propuesta. Seguidamente, en la sección 2.2, se exponen los cálculos de procesos más representativos de la IS, haciendo énfasis en el cálculo temporizado que nos permiten formalizar los aspectos temporales de BPMN. Luego, en la sección 2.3, presentamos algunos aspectos de la verificación formal como preámbulo a la sección 2.4, en la cual se ahonda en la verificación automática. A continuación, en la sección 2.5.5, se describen los elementos que giran alrededor de la verificación composicional, ya que ésta constituye la base fundamental de nuestro enfoque de verificación. Finalmente, en la sección 2.6 exponemos los aspectos de los BP que guardan relación directa con nuestra propuesta y describimos brevemente los formalismos más conocidos utilizados en el BPM.

2.1 Especificación formal de propiedades – lógicas temporales

Las lógicas temporales permiten especificar el comportamiento de los sistemas reactivos [49], sin introducir el tiempo explícitamente. Describen como los componentes, protocolos, objetos, módulos, procedimientos y funciones, se comportan con el paso del tiempo. Se definen como la formalización de enunciados que incluyan precisiones acerca del tiempo en los cuales ocurren [22]. Las lógicas temporales se basan en las transiciones entre estados del sistema; es decir, de las secuencias de computación realizadas por el sistema. Un sistema deben ser modelado entonces como un sistema de transición o *Estructura de Kripke* (EK) (ver Definición 2.10) con una función que etiqueta cada estado con las proposiones de la lógica temporal que se mantienen en ese estado [22].

2.1.1 Estructuras de Kripke

Una EK es un tipo de máquina de estados finitos no determinista propuesto por S. Kripke en 1963. Una EK es un formalismo de modelado independiente de una forma particular de representar el comportamiento de un sistema. Estas estructuras son básicamente grafos que contienen los estados alcanzables por el sistema en los nodos, y los arcos contituyen las transiciones de estados del sistema [11, 22]. Las lógicas temporales son tradicionalmente definidas en términos de EK, ya que permiten establecer lo que se espera del sistema al alcanzar sus estados. Contienen un etiquetado de los estados del sistema con las Proposiciones Atómicas (*Atomic Propositions*, AP) que se satisfacen en cada estado, según la Definición 2.10.

Definición 2.1. Estructura de Kripke. Sea AP un conjunto no vacío de proposiciones atómicas. Una *Estructura de Kripke* (EK) es una tupla $M = \langle S, s_0, R, L \rangle$, donde:

- S es un conjunto finito de estados,
- $s_0 \in S$ es un estado inicial,
- $R \subseteq S \times S$ es una relación de transición, para la que se mantiene que $\forall\, s \in S : \exists\, s' \in S : (s, s') \in R$, y
- $L : S \to 2^{AP}$ es el etiquetado, una función que etiqueta cada estado con las AP que se mantienen en ese estado.

Para obtener una EK que formule un *grafo de alcanzabilidad* (es decir, *el grafo constituido por los estados y las transiciones alcanzables por el sistema*) se necesita primero fijar un conjunto de AP, las cuales denotan las propiedades que interesa se alcancen en los estados individuales. El etiquetado de los estados (con marcaje) del grafo de alcanzabilidad se sustituye por el etiquetado que muestra que las proposiciones atómicas se mantienen en ese estado. Después, las etiquetas (con las transiciones) en los arcos del grafo de alcanzabilidad son eliminados, y el resultado es la EK[1].

A continuación se describen las lógicas temporales más conocidas y que sustentan la lógica temporal que proponemos usar en nuestro enfoque de verificación.

[1] La eliminación de las etiquetas de los arcos podría resultar en una fusión de algunos de los nodos.

*2.1.2 Lógica de Árbol Computacional**

La Lógica de Árbol Computacional (*Computational Tree Logic*, CTL*) especifica propiedades del árbol de computación de un sistema de transición [16, 22, 49]. Este árbol (infinito) se obtiene designando a un estado del sistema como la raíz (estado inicial), y muestra todas las posibles computaciones desde ese estado. CTL* está compuesta por operadores temporales y de cuantificación sobre caminos:

- La cuantificación sobre caminos está dada por los operadores A (para todo camino) y E (existe un camino). Estos cuantificadores son utilizados en un estado particular del sistema de transición y evaluadas sobre el árbol de computación relacionado con ese estado.
- Los operadores temporales sirven para describir propiedades de un camino en el árbol de computación. Se usan 4 operadores básicos:
  - Unarios: X (proximo estado), F (en el futuro), y G (siempre en el futuro).
  - Binario: U (algo ocurre hasta que otra cosa ocurra).

La sintáxis de CTL* consiste de fórmulas de estado (que se evalúan en un estado del sistema) y de camino (que se evalúan en una rama del árbol), tal como se indica en la defición 2.2.

Definición 2.2. Sintaxis de CTL*. Dadas las fórmulas de estado ϕ y ϕ' y las fórmulas de camino ψ y ψ', sobre un alfabeto Σ, la *sintaxis de CTL** es como sigue:

- Si $a \in \Sigma$, entonces $\phi(a)$ es una fórmula de estado.
- Si ϕ y ϕ' son fórmulas de estado, entonces $\neg\phi$, $\phi \wedge \phi'$ y $\phi \vee \phi'$ son fórmulas de estado.
- Toda fórmula de estado es también una fórmula de camino.
- Si ψ es una fórmula de camino, entonces $\mathsf{E}\psi$ y $\mathsf{A}\psi$ son fórmulas de camino.
- Si ψ y ψ' son fórmulas de camino, entonces $\neg\psi$, $\psi \wedge \psi'$ y $\psi \vee \psi'$ son formulas de camino.
- Si ψ y ψ' son fórmulas de camino, entonces $\mathsf{X}\psi$, $\mathsf{F}\psi$, $\mathsf{G}\psi$ y $\psi\mathsf{U}\psi'$ son fórmulas de camino.

Para definir la semántica de CTL* debemos conocer primero qué es un camino en un sistema de transición de estados o EK. Sea M un sistema de transición, un camino π sobre M es una secuencia $s_0, s_1, \ldots$ tal que para cada $i \geq 0$, $(s_i, s_{i+1}) \in R$; es decir, el camino $s_0, s_1, \ldots$ es una rama del

árbol de computación de M con raíz s_0. Denotamos por π^i el sufijo de π que empieza en s_i. En la Definición 2.3 se sintetiza la semántica de CTL*.

Definición 2.3. Semántica de CTL*. Dado un sistema de transición M, un estado s, un camino π, unas fórmulas de estado ϕ y ϕ', y unas fórmulas de camino ψ y ψ', la *semántica de CTL** es como sigue::

$(M,s) \models a$	si y sólo si $s \in AP_a$
$(M,s) \models \neg\phi$	si y sólo si no es el caso que $(M,s) \models \phi$
$(M,s) \models \phi \vee \phi'$	si y sólo si $(M,s) \models \phi$ o $(M,s) \models \phi'$
$(M,s) \models \phi \wedge \phi'$	si y sólo si $(M,s) \models \phi$ y $(M,s) \models \phi'$
$(M,s) \models \mathsf{E}\psi$	si y sólo si existe un camino π de la forma $s\ s_1 \ldots$ tal que $(M,\pi) \models \psi$
$(M,s) \models \mathsf{A}\psi$	si y sólo si para todo camino π de la forma $s\ s_1 \ldots$ se tiene que $(M,\pi) \models \psi$
$(M,\pi) \models \psi$	si y sólo si s es el primer estado de π y $(M,s) \models \psi$
$(M,\pi) \models \mathsf{X}\psi$	si y sólo si $(M,\pi^1) \models \psi$
$(M,\pi) \models \mathsf{F}\psi$	si y sólo si existe $j \geq 0$ tal que $(M,\pi^j) \models \psi$
$(M,\pi) \models \mathsf{G}\psi$	si y sólo si para todo $j \geq 0$ se tiene que $(M,\pi^j) \models \psi$
$(M,\pi) \models \psi\mathsf{U}\psi'$	si y sólo si existe $j \geq 0$ tal que $(M,\pi^j) \models \psi'$ y para todo $0 \leq k < j$ se tiene que $(M,\pi^k) \models \psi$
Cualquier combinación Booleana de fórmulas de camino evaluada de forma estándar	

Con CTL* se pueden expresar propiedades tales como *inevitablemente sucederá el evento e* o *la propiedad p se satisface continuamente en todas las ejecuciones del sistema*.

2.1.3 Lógica Temporal Lineal

La Lógica Temporal Lineal (*Linear Temporal Logic*, LTL) es el conjunto de fórmulas de camino de CTL* que se construyen sólo a partir de las AP. Es decir, LTL es la clase de fórmulas dadas por la gramática indicada en la Definición 2.4.

Definición 2.4. Gramática de LTL. Dadas las fórmulas de camino ψ y ψ', sobre un alfabeto Σ ($a \in \Sigma$), la *gramática de LTL* es como sigue:

$$\psi, \psi' := a \mid \neg\psi \mid \psi \wedge \psi' \mid \psi \vee \psi' \mid \mathsf{X}\psi \mid \psi\mathsf{U}\psi'$$

Las fórmulas en LTL se evalúan de forma natural sobre secuencias $s_0, s_1, \ldots$ de elementos en Σ. Para evaluar una fórmula en LTL en un estado s de un sistema de transición M:

$$(M, s) \models \psi \text{ si y sólo si } (M, s) \models \mathsf{A}\psi$$

2.1.4 Lógica de Árbol Computacional Temporizada

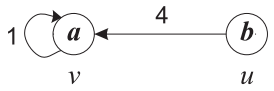
En nuestra propuesta, utilizamos la Lógica de Árbol Computacional Temporizada (*Clocked Computation Tree Logic*, CCTL) para especificar las propiedades que los BP deben cumplir y que son utilizadas para su posterior verificación. A continuación se describe esta lógica temporal, comenzando por la interpretación que se les da a sus operadores según el contexto temporal de las estructuras de intervalo. Los conceptos que se presentan brevemente en esta sección pueden consultarse en [188].

2.1.4.1 Estructuras de intervalo

Un intervalo está formado por la identificación de sus extremos, los cuales son instantes que satisfacen ciertas propiedades. Estos puntos representan la interpretación contextual de los operadores temporales dentro de una secuencia infinita de estados que corresponden a una ejecución de un sistema. Todos los operadores de intervalo pueden ser afectados por un límite de tiempo único; es decir, $\mathsf{Quantor\_Operator}_{[instant]}$*Nested_Formula*. Una vez que los extremos de cierto intervalo ha sido localizado o el límite inferior de un intervalo es fijado a cero, en el caso de existir un único límite temporal, la semántica de la fórmula que lo acompaña (*Nested_Formula*) restringue su interpretación al contexto de tiempo discreto dado por un subconjunto de estados sobre una ruta de ejecución. Es decir, $\mathsf{EF}_{[5]}\varphi$ significa que existe un camino de tal manera que φ se mantiene en uno de los siguientes cinco estados.

En consecuencia, cada intervalo representa un contexto temporal específico, el cual es un subconjunto de estados del contexto global, sobre el cual las fórmulas temporales alcanzan una interpretación diferente con respecto a cuando no se especifica el intervalo. Consideremos la *estructura temporizada* [187] de la Fig. 2.1, con sólo dos estados $\{v, u\}$, sobre los cuales las proposiciones (a, b) deben mantenerse, respectivamente, y la única transición entre estos está etiquetada con un retraso de tiempo igual a 4 unidades. Entonces, se mantienen los conjuntos de estados que se indican para las tres fórmulas de la Fig. 2.1.

Las lógicas de intervalo nos permiten llevar a cabo un razonamiento lógico a nivel de intervalos de tiempo, en lugar de instantes. Dentro de nues-



- $\mathsf{EF}[3]a \equiv \{v\}$
- $\mathsf{EF}[3]\mathsf{EF}[3]a \equiv \{v\}$
- $\mathsf{EF}[6]a \equiv \{u, v\}$

Fig. 2.1 Ejemplo de una estructura temporizada.

tro enfoque, el modelo básico para entender los sistemas con criticidad es la *Estructura de Intervalo* (EI) [188]; es decir, un sistema de transición de estados con transiciones etiquetadas, asumiendo que cada estructura de intervalo tiene exactamente un reloj para medir el tiempo, según la Definición 2.5.

Definición 2.5. Estructura de intervalo. Una *Estructura de Intervalo* (EI) es una tupla $\mathbf{I} = \langle AP, S, T, L, I \rangle$ con un conjunto de proposiciones atómicas AP, un conjunto de estados S, un conjunto de estados iniciales I, una función $T : S \times S \to \wp^{\omega}(\mathbb{N}_0)$ que conecta a los estados con transiciones etiquetadas, y una función de etiquetado de estados $L : S \to \wp(AP)$.

Ahora asociamos los estados con los valores de reloj, de acuerdo a la Definición 2.6.

Definición 2.6. Configuración. Una *configuración* es un par $g = (s, \nu) \in S \times \mathbb{N}_0$ donde el estado s es asociado con un posible valor de reloj ν. El conjunto de todas las configuraciones es:

$$G = \{(s, \nu) \mid (s, \nu) \in S \times \mathbb{N}_0 \wedge 0 \leq \nu \leq max\_time(s)\} \tag{2.1}$$

Una interpretación de reloj ν es un mapeo de Cl a $\mathbb{N}_0$ que asigna un valor natural $n \in \mathbb{N}_0$ para cada reloj del conjunto de relojes Cl. Decimos que una interpretación de reloj ν para Cl satisface una restricción de reloj δ sobre Cl si y sólo si δ evalúa verdadero usando los valores dados por ν. Para $t \in \mathbb{N}_0$, $\nu + t$ denota la interpretación de reloj que mapea cada reloj x al valor $\nu(x) + t$, y la interpretación de reloj $t \cdot \nu$ asigna a cada reloj x el valor $t \cdot \nu(x)$. Para $Y \subseteq Cl$, $[Y \mapsto t]\nu$ denota la interpretación de reloj para Cl la cual asigna t para cada $x \in Y$, y de acuerdo con ν sobre el resto de los relojes.

2.1.4.2 Sintaxis y semántica

CCTL [187, 186] se usa para razonar con secuencias de estados, donde un estado es una interpretación cierta que asigna valores verdaderos a proposiciones atómicas del lenguaje, y el tiempo es isomorfo al conjunto de enteros no negativos. CCTL incluye los operadores *Until* (U) y *Next* (X) de CTL [46] y otros operadores derivados de LTL, tal como *Release* (R), *Weak Un-*

til (W), *Cancel* (C) y *Since* (S), que se usan para facilitar la especificación de sistemas con criticidad (ver la Definición 2.7. Todos los operadores temporales están precedidos por un cuantificador (A, universal; E, existential) el cual determina si el operador temporal debe ser interpretado sobre una ejecución (cuantificación existencial) o sobre cada ejecución (cuantificación universal) a partir de la configuración actual. Ver [188] para más detalles.

Definición 2.7. Sintaxis de CCTL. Dada una proposición atómica $p \in AP$, las fórmulas arbitrarias CCTL φ y ψ, y los límites de tiempo $a \in \mathbb{N}_0$ y $b \in \mathbb{N}_0 \cup \{\infty\}$, la *sintaxis de CCTL* está definida por:

$$\varphi, \psi := \begin{cases} p \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \rightarrow \psi \mid \varphi \leftrightarrow \psi \\ \mid \mathsf{EX}_{[a]}\varphi \mid \mathsf{EF}_{[a,b]}\varphi \mid \mathsf{EG}_{[a,b]}\varphi \mid \mathsf{E}(\varphi \mathsf{U}_{[a,b]}\psi) \\ \mid \mathsf{E}(\varphi \mathsf{R}_{[a,b]}\psi) \mid \mathsf{E}(\varphi \mathsf{B}_{[a,b]}\psi) \mid \mathsf{E}(\varphi \mathsf{C}_{[a]}\psi) \mid \mathsf{E}(\varphi \mathsf{S}_{[a]}\psi) \\ \mid \mathsf{AX}_{[a]}\varphi \mid \mathsf{AF}_{[a,b]}\varphi \mid \mathsf{AG}_{[a,b]}\varphi \mid \mathsf{A}(\varphi \mathsf{U}_{[a,b]}\psi) \\ \mid \mathsf{A}(\varphi \mathsf{R}_{[a,b]}\psi) \mid \mathsf{A}(\varphi \mathsf{B}_{[a,b]}\psi) \mid \mathsf{A}(\varphi \mathsf{C}_{[a]}\psi) \mid \mathsf{A}(\varphi \mathsf{S}_{[a]}\psi) \end{cases} \tag{2.2}$$

No obstante, y a diferencia de otras lógicas temporales —como *Metric and Layered Temporal Logic* (MLTL) [161] o *Timed Computation Tree Logic* (TCTL) [97]), la semántica de los operadores temporales de CCTL es interpretada sobre intervalos conformados por una serie de instantes de tiempo, los cuales son dados por las relaciones de validación presentadas en la Definición 2.8:

Definición 2.8. Semántica de CCTL. Dada la EI $\mathbf{I} = \langle AP, S, T, L, I \rangle$, la configuración $g_0 = (s, v) \in G$, el función de etiquetado $L(s)$, y los límites de tiempo $a \in \mathbb{N}_0$ y $b \in \mathbb{N}_0 \cup \{\infty\}$, la *semántica de CCTL* se deriva de las siguientes relaciones:

$$\begin{array}{ll}
(\mathbf{I}, g_0) \models p & :\equiv p \in L(s) \\
(\mathbf{I}, g_0) \models \neg\varphi & :\equiv \text{ no } (\mathbf{I}, g_0) \models \varphi \\
(\mathbf{I}, g_0) \models (\varphi \wedge \psi) & :\equiv (\mathbf{I}, g_0) \models \varphi \text{ y } (\mathbf{I}, g_0) \models \psi \\
(\mathbf{I}, g_0) \models \mathsf{EX}_{[a]}\varphi & :\equiv \text{existe una ejecución } r = (g_0, \dots) \text{ tal que} \\
 & \quad \exists\ i > a \text{ donde se mantiene } (\mathbf{I}, g_i) \models \varphi \\
(\mathbf{I}, g_0) \models \mathsf{EF}_{[a,b]}\varphi & :\equiv \text{existe una ejecución } r = (g_0, \dots) \text{ tal que} \\
 & \quad \exists\ i \mid a \leq i \leq b \text{ donde se mantiene } (\mathbf{I}, g_i) \models \varphi \\
(\mathbf{I}, g_0) \models \mathsf{EG}_{[a,b]}\varphi & :\equiv \text{existe una ejecución } r = (g_0, \dots) \text{ tal que} \\
 & \quad \forall\ i \mid a \leq i \leq b \text{ donde se mantiene } (\mathbf{I}, g_i) \models \varphi \\
(\mathbf{I}, g_0) \models \mathsf{E}(\varphi \mathsf{U}_{[a,b]}\psi) & :\equiv \text{existe una ejecución } r = (g_0, \dots) \text{ y un } a \leq i \leq b \\
 & \quad \text{tal que } g_i \models \psi \text{ y } \forall j < i \text{ donde se} \\
 & \quad \text{mantiene } (\mathbf{I}, g_j) \models \varphi
\end{array} \tag{2.3}$$

Los otros operadores pueden ser derivados por distintas definiciones:

$$
\begin{aligned}
(\mathbf{I}, g_0) &\models \mathsf{AX}_{[a]}\varphi \quad &&:\equiv \neg\mathsf{EX}_{[a]}\neg\varphi \\
(\mathbf{I}, g_0) &\models \mathsf{AF}_{[a,b]}\varphi \quad &&:\equiv \neg\mathsf{EG}_{[a,b]}\neg\varphi \\
(\mathbf{I}, g_0) &\models \mathsf{AG}_{[a,b]}\varphi \quad &&:\equiv \neg\mathsf{EF}_{[a,b]}\neg\varphi \\
(\mathbf{I}, g_0) &\models \mathsf{A}(\varphi\mathsf{U}_{[a,b]}\psi) \quad &&:\equiv \neg\mathsf{E}[\neg\psi\mathsf{U}_{[a,b]}(\neg\varphi \wedge \neg\psi)] \wedge \neg\mathsf{EG}_{[a,b]}\neg\psi
\end{aligned}
\tag{2.4}
$$

En la Tabla 2.1 se puede ver una descripción textual de los operadores temporales utilizados usualmente en especificaciones CCTL y que trata de ilustrar su correcta interpretación.

$\mathsf{X}_{[a]}\varphi$	La fórmula φ tiene que mantenerse después de exactamente a unidades de tiempo.
$\mathsf{F}_{[a,b]}\varphi$	La fórmula φ tiene que mantenerse al menos una vez dentro del intervalo $[a,b]$.
$\mathsf{G}_{[a,b]}\varphi$	La fórmula φ tiene que mantenerse todo el tiempo del intervalo $[a,b]$.
$\varphi\mathsf{U}_{[a,b]}\psi$	La fórmula ψ tiene que hacerse verdadera dentro del intervalo $[a,b]$ y todos los pasos de tiempo anteriores, la fórmula φ tiene que ser válida.
$\varphi\mathsf{R}_{[a,b]}\psi$	Es la lógica dual del operador U. La fórmula ψ tiene que hacerse verdad a lo largo del intervalo $[a,b]$ incluyendo la primera instancia cuando la formula φ debe de ser válida. Sin embargo, no se requiere que la fórmula φ se mantenga eventualmente.
$\varphi\mathsf{B}_{[a,b]}\psi$	Si ψ se hace verdadera durante el intervalo $[a,b]$, entonces φ tiene que ser válida una instancia de tiempo antes de este evento. De otra manera φ tiene que ser válida al menos una vez hasta el tiempo b.
$\varphi\mathsf{C}_{[a]}\psi$	Si la fórmula φ es verdadera en la ejecución actual hasta el tiempo $a-1$ entonces la fórmula ψ tiene que mantenerse hasta el tiempo a.
$\varphi\mathsf{S}_{[a]}\psi$	Desde el tiempo cero y hasta el tiempo $a-1$ la fórmula φ tiene que mantenerse y en el tiempo a la fórmula ψ tiene que hacerse verdadera.

Tabla 2.1 Descripción informal de operadores temporales.

Revisadas las lógicas temporales más conocidas a la fecha y estudiada la que usaremos en nuestro enfoque, en la próxima sección trataremos los aspectos más resaltantes de la especicación formal del comportamiento de sistemas.

2.2 Especificación formal del comportamiento de sistemas

La concurrencia inherente a los sistemas con criticidad o reactivos [87] se consigue dividiendo el sistema en un conjunto de procesos independientes funcionando en paralelo, compartiendo información mediante alguno de los mecanismos de comunicación entre procesos que proporcione el lenguaje de modelado. Adicionalmente, en un sistema con criticidad las únicas entidades observables son los *eventos* [7, 130]. Los eventos son producidos por el entorno para comunicar cosas al sistema y éste emite eventos al entorno como respuesta a los anteriores. Asimismo, los eventos se utilizan para comunicar sucesos entre los procesos en los cuales está definido el sistema reactivo

[7, 130]. Bajo las premisas anteriores, los *cálculos de proceso* (también conocidos como *álgebras de proceso* [7]) son un potente formalismo para especificar sistemas dinámicos [14]. Son completamente formales basados en teorías algebraicas e integran diversos métodos de verificación [7].

2.2.1 Redes de Petri

Las *Redes de Petri* (*Petri Nets*, PN) son grafos dirigidos bipartitos con un estado inicial llamado *marcación inicial*. Los dos componentes principales de las PN son los *sitios* (también conocidos como *estados*) y las *transiciones* [155]. Gráficamente, los sitios son dibujados como círculos y las transiciones como barras o rectángulos. Las aristas del grafo son conocidas como *arcos*, los cuales tienen un peso específico, indicado por un número entero positivo, y van de sitio a transición y viceversa. Por simplicidad, el peso de los arcos no se indica cuando éste es igual a 1. Un arco que esté etiquetado con k puede ser interpretado como k arcos paralelos [41, 122]. La Definición 2.9 establece formalmente la relación entre los componentes de una PN.

Definición 2.9. Red de Petri. Una *Red de Petri* (PN) es una tupla PN = $\langle P, T, F, W, M_0 \rangle$ donde:

- $P = \{p_1, p_2, \ldots, p_m\}$ es un conjunto finito de sitios.
- $T = \{t_1, t_2, \ldots, t_n\}$ es un conjunto finito de transiciones.
- $F \subseteq (P \times T) \cup (T \times P)$ es un conjunto de arcos.
- $W : F \to \mathbb{Z}^+$ es una función de peso.
- $M_0 : P \to \mathbb{Z}_0^+$ es la marcación inicial.
- $P \cap T = \varnothing$ y $P \cup T \neq \varnothing$.

donde, para una transición t_i, los sitios de entrada y los sitios de salida serán denotados como $\bullet t_i = \{p_i | (p_i, t_i) \in F\}$ y $t_i \bullet = \{p_i | (t_i, p_i) \in F\}$, respectivamente; y una transición t_i está habilitada con una marcación M_i si cada sitio de entrada p_i está marcado con al menos $W_i(p_i, t_i)$ tokens.

El estado del sistema que la red esté modelando es representado con la asignación de enteros no–negativos a los sitios. Esta asignación es conocida como una *marcación*, la cual es representada gráficamente mediante unos pequeños círculos negros dentro de un sitio p, llamados *tokens*. Si el número de tokens es demasiado grande, los k tokens son representados con un número no–negativo dentro del correspondiente sitio [155].

Típicamente, los estados representan algún tipo de condición en el sistema, y una transición representa un evento. Un sitio de entrada (salida) a una

transición representan las *pre–* (*post–*) *condiciones* [41]. Los tokens pueden tener muchas interpretaciones. Por ejemplo, cuando un sitio está marcado con un token, este puede representar que la correspondiente condición es verdadera. Debido a que las redes de Petri pueden modelar muchos tipos de sistemas, lo que representen los sitios, transiciones y tokens, varía enormemente [122].

Una transición puede o no ser disparada al estar habilitada. Cuando más de una transición es habilitada, alguna de ellas es seleccionada de manera no–determinística dependiendo del modelo empleado [122]. Conforme las transiciones son disparadas, el número total de tokens distribuidos a lo largo de la red puede variar, esto es, la conservación de los tokens no siempre sucede. Según [41], las PN son una herramienta que permite el modelado de sistemas que son considerados no–deterministas, concurrentes, paralelos, asíncronos, distribuidos y/o estocásticos. La principal fortaleza de las PN es su soporte para el análisis de propiedades y problemas asociados con sistemas concurrentes relacionados con el control del flujo.

2.2.2 Cálculos de procesos

Los *cálculos de proceso* son lenguajes abstractos de especificación donde las ideas de proceso e interacción son centrales en la programación de sistemas [7, 14, 127]. Están definidos en términos de operaciones primitivas que establecen el tipo de interacciones posibles entre los procesos, así como su evolución en el tiempo. Este conjunto de operaciones promueve una construcción composicional de las especificaciones: *la definición formal de un sistema tiene lugar por la composición de procesos simples que representan los componentes que lo conforman* [7, 14, 127].

Estas características permiten considerar los cálculos de proceso como instrumentos de abstracción de sistemas concurrentes complejos [7]. Esto significa que el modelado de un sistema utilizando cálculos de proceso se concentra en ciertos aspectos esenciales que determinan su comportamiento, mientras que otros, menos importantes de acuerdo con algún criterio particular, no son considerados. Este estilo de especificación, además, facilita enormemente la verificación de propiedades esenciales de modelos de los sistemas [14, 127, 190]. Seguidamente haremos una introducción a los cálculos de proceso más importantes, profundizando en el que utilizamos en la propuesta que hacemos en este trabajo.

2.2.2.1 Π–calculus

El *Π–calculus* es un lenguaje formal que es parte de la familia de los cálculos de proceso, cuyo objetivo es la descripción y el análisis de propiedades del procesamiento concurrente [41]. Fue desarrollado por Milner [148], como una interpretación de los procesos concurrentes, dinámicos (o llamados también *móviles*) y su comunicación; por lo tanto, esta herramienta provee una infraestructura para la representación, simulación, análisis y verificación de sistemas dinámicos de comunicación [148].

Un sistema representado en Π–calculus consta de múltiples procesos concurrentes, de los cuales hay parejas de procesos que interactúan entre ellos, enviando y recibiendo mensajes de manera sincronizada. Esta comunicación es realizada sobre canales complementarios de entrada y salida. Así, cuando un proceso receptor recibe un mensaje, recibe también un canal, el que puede utilizar para comunicar sus respuestas. Esta característica, llamada *movilidad*, permite que las conexiones de la red cambien con las interacciones realizadas; es decir, se producen cambios dinámicos en la topología de la comunicación entre procesos [148]. La sintaxis de Π–calculus se resume en la Definición 2.10.

Definición 2.10. Sintaxis de Π–calculus. Sean los procesos P y Q, el nombre x y el prefijo Π. El conjunto $\mathbf{P}^{\Pi}$ de expresiones de proceso de Π–calculus se define mediante la siguiente *sintaxis*:

$$P, Q ::= P \| Q \mid !P \mid (\nu x)P \mid \Sigma_{i \in I} \Pi_i \cdot P_i$$

donde el prefijo Π denota una acción atómica previa a un proceso, según:

$$\Pi ::= x(y) \mid \overline{x}\langle y \rangle \mid \tau$$

de acuerdo a:

- $x(y)$ denota la recepción de un dato y por el canal x.
- $\overline{x}\langle y \rangle$ denota el envío de un dato y por el canal x.
- τ denota una acción no observable.

El proceso $P \| Q$ denota la *composición paralela* de los subprocesos P y Q. Además, permite que P y Q se comuniquen. El proceso $!P$ representa la *replicación* de P. Es decir, define la ejecución infinita de P. Un *nombre* puede estar restringido al contexto de un proceso particular. Este es el propósito de $(\nu x)P$, en donde el nombre x es local a P y solamente visible dentro de él. Finalmente, la selección no determinística entre una serie de procesos $\Pi_i \cdot P_i$ $(i \in I)$ se representa como su sumatoria $\Sigma_{i \in I} \Pi_i \cdot P_i$. Esta selección depende de la comunicación de dichos procesos. Dos pro-

cesos importantes pueden derivarse de la sumatoria: 0 y $\Pi \cdot P$. El primero representa la acción nula y es la abreviación de la sumatoria cuando $I = \varnothing$, mientras que el segundo se da cuando sólo hay un proceso involucrado en la selección.

2.2.2.2 Cálculo de Sistemas de Comunicación

El formalismo Cálculo de Sistemas de Comunicación (*Calculus of Communicating Systems*, CCS) [147] es un lenguaje de especificación basado en el cálculo de procesos para la especificación y modelado de sistemas discretos comunicantes. El lenguaje CCS fue propuesto por Milner [147] para ejemplificar su idea de un cálculo para representar simbólicamente los procesos que conforman un sistema de software paralelo, su proposición fue hecha poco antes que la de CSP de Hoare [92], formando ambos lenguajes los ejemplos por excelencia de lo que es un cálculo de procesos. De acuerdo con [127], la principal diferencia entre CSP y CCS se encuentra en lo que se considera como la representación matemática adecuada de la noción de proceso: un proceso en CSP está representado matemáticamente como un conjunto de *fallos*[2], mientras que en CCS éstos son un elemento del conjunto del sistema de etiquetado de transiciones equivalente al módulo de observación.

CCS propone una notación textual y otra visual para representar la existencia dentro de un sistema de lo que llama proceso y la definición de éstos [147]. Los procesos son vistos como bloques herméticos que comunican con el mundo externo o entorno por medio de puertos bien específicos, que conforman lo que se conoce como interfaz del proceso. Los procesos definen su comportamiento enunciando explícitamente la secuencia entera de operaciones elementales que dicho proceso efectúa durante toda su existencia [147].

2.2.2.3 Álgebra de Procesos de Comunicación

Según [127], un verdadero enfoque algebraico abstracto, conocido como Álgebra de Procesos de Comunicación (*Algebra of Communicating Processes*, ACP), fue propuesto por primera vez explícitamente en [23, 24]. La propuesta de Bergstra y Klop [23, 24] estudia la teoría de procesos sin recurrir a una definición matemática de *proceso*; de la misma manera como la teoría matemática de los anillos es la aritmética sin recurrir a una definición

[2] Un *fallo* es la secuencia de eventos en los cuales un proceso puede participar junto con una serie de eventos en los cuales posteriormente el proceso se niega a realizar [92, 127, 184, 190].

matemática de *número* [127]. Hay una fuerte relación entre ACP con el cálculo CCS de Milner. En [147] se discuten algunos ámbitos de proceso que puede ser vistos como modelos de ACP. Determinar la relación precisa es cuestión de una investigación detallada [14, 127]. Preliminarmente se puede decir que ACP es una formulación abstracta de CCS [127]. En particular, los operadores de ACP distintos a los de CCS son [23]:

1. la multiplicación ($\cdot$) es general (no sólo multiplicación de prefijo),
2. NIL está ausente en ACP,
3. δ, $\between$, y $\mid$ no están presentes en CCS.

El operador *merge* ($\|$) es el mismo que en CCS, aunque es axiomatizado diferente. ACP no tiene operadores de reetiquetado explícitos como en CCS, o *morfismos*, como se les llama en [147], con excepción de los operadores de encapsulación ∂_H que desempeñan el rol de *restricción* en CCS. También en ACP no se tiene τ–pasos (pasos de silencio) y tampoco leyes τ para ellas [147]; se pueden añadir sistemáticamente, e incluso de forma conservadora, a ACP. El sistema de axiomas derivados ACP_τ son estudiados en [23]. En general, ACP no aborda el complicado problema de *ocultar* o hacer abstracción en los procesos [24].

2.2.2.4 Procesos Secuenciales de Comunicación

El formalismo Procesos Secuenciales de Comunicación (*Communicating Sequential Processes*, CSP) fue propuesto como un marco teórico–práctico dentro del cual se puede estudiar formalmente el problema de la concurrencia y dominar su complejidad [184, 190]. Los procesos se construyen combinando eventos y otros procesos por medio de operadores, formándose así un cáculo de procesos.

CSP [92, 184] nació como una notación de programación para sistemas distribuidos y paralelos que ha dado lugar a varios lenguajes de programación. Es una notación basada en la ocurrencia de eventos —señales y comunicaciones, entre otros—, fundamentalmente orientada a la descripción de secuencias de eventos que ocurren asociados al comportamiento de un proceso y a la sincronización (o comunicación) necesaria entre procesos. CSP constituye un paradigma cómodo para llevar a cabo la programación de multiprocesadores de memoria distribuida y posee ventajas adicionales, tales como propiciar la capacidad de verificación y la portabilidad entre diferentes arquitecturas de los programas que se obtienen utilizando un lenguaje que tenga como base a esta notación. CSP está basado en un cálculo teórico que proporciona un conjunto de términos de procesos matemáticamente bien definidos (procesos concurrentes válidos) y derivados de una gramática abs-

tracta que incluye operadores para expresar la composición no determinista, concurrente de estos términos. Según [14], existen al menos tres formas de interpretación semántica de CSP:

1. *Operacional*. Los procesos CSP se explican en términos de máquinas de estados infinitos. Es la forma más cercana a una implementación y la que menos conocimientos de matemática requiere, pero a la vez puede ser engorroso formalizar algunos conceptos.
2. *Denotacional*. CSP se basa en teorías matemáticas sólidas como lo son la teoría de conjuntos y la matemática discreta. Cada elemento del lenguaje se explica en términos de esas teorías y luego las construcciones compuestas de CSP se componen en la teoría con los elementos propios de la teoría.
3. *Axiomática*. Esta es una forma especialmente adecuada para dar la semántica de un lenguaje como CSP. Se procede como en matemática: *se axiomatizan ciertas propiedades y de esta axiomatización se deducen teoremas sobre la teoría*. Las propiedades toman la forma de leyes algebraicas de ciertos operadores. Estos axiomas son los teoremas que se deducen si la semántica se expresa de forma denotacional, de aquí la relación entre ambas formas. La forma algebraica puede considerarse como la manera más abstracta de formalizar el lenguaje.

Según [184], CSP está constituido por dos lenguajes: el núcleo del formalismo formado por eventos, operadores y procesos; y un lenguaje para especificar eventos compuestos, parámetros, canales y procesos parametrizados.

La notación de programación CSP está basada en órdenes que especifican el comportamiento de un componente que lo ejecuta y su resultado puede tener éxito o fallos. Si la ejecución de una única orden tiene éxito, puede tener efecto sobre el estado interno del componente, esto es, la orden de asignación puede modificar los valores de las variables que representan el estado del componente, o bien en el entorno del componente (orden de salida), o bien en ambos (orden que representa la recepción de un mensaje y su asignación a una variable objetivo del proceso receptor). La ejecución de un proceso estructurado implica la ejecución de uno o de todas sus órdenes constituyentes; dependiendo de su tipo pueden ser [92, 184, 190]: composición concurrente ($\|$), orden alternativa ($|$), repetitiva (*), guardas ($\rightarrow$), elección externa ($\Box$) (selección determinista de una guarda abierta dentro de una orden alternativa y estructurada), elección interna ($\sqcap$) (selección no determinista), ocultación de conjuntos de acciones de comunicación ($\setminus$) y renombrado ($[A \mapsto B]$) de conjuntos de símbolos sintácticos.

Así mismo, se puede especificar tanto la composición secuencial de un evento y un proceso, a través de $a \rightarrow P$, como la composición secuencial

de procesos, de la forma $P \fatsemi Q$. Además, se utilizan acciones distintas para representar la sincronización y la comunicación entre procesos. En CSP la sincronización de dos procesos tiene lugar cuando se indica un evento sin signo en la especificación del proceso. Por ejemplo, siendo P y Q procesos y a un identificador de un evento genérico, los procesos $a \rightarrow P$ y $a \rightarrow Q$, si se componen concurrentemente (usando el operador $\|$), han de sincronizarse en el evento a, del que, como se puede observar, no se indica signo. Por el contrario, la comunicación entre procesos implica intercambio de datos, por lo que se debe indicar que uno de los procesos realiza una acción de salida y el otro la acción de entrada complementaria. De este modo, si P y Q son procesos, c es un canal y x e y son valores, los procesos $c!x \rightarrow P$ y $c?y \rightarrow Q$ pueden comunicarse a través de dicho canal, enviando el primero de ellos el valor x al segundo que instancia el valor de la variable objetivo y al recibirlo.

En CSP se distinguen tres formas de terminación al especificar un proceso:

- *STOP* modela las situaciones de interbloqueo, es decir, representa a un proceso que no es capaz de realizar ninguna acción.
- *SKIP* modela la terminación con éxito, para lo cual se considera que el proceso realiza una acción especial '$\surd$' y acaba.
- *RUN* representa a un proceso divergente, es decir, que es capaz de realizar indefinidamente cualquier acción y que, por lo tanto, nunca acaba pero que ya no se comunicará más con ningún otro proceso de su entorno.

Por su parte, al componer procesos en paralelo, esta composición puede definirse como un *entrelazado* (en inglés, *interleaving*) de las acciones de ambos procesos (lo que se indica mediante el operador $|||$), o como una composición concurrente de las acciones que son susceptibles de sincronización (mediante el operador $\|$), en cuyo caso se impone la sincronización de los eventos comunes de ambos procesos. En síntesis, el cálculo CSP es definido por reglas gramaticales [59] indicadas en la definición 2.11.

Definición 2.11. Sintaxis de CSP. Dado un conjunto Σ de símbolos, que pueden ser compuestos con un conjunto de canales de comunicación **C** usando el operador punto (.), y un conjunto de nombres de procesos **P**, la *sintaxis de CSP* es dada por:

$$P, Q ::= \begin{cases} \mathit{STOP} \mid \mathit{SKIP} \mid a \rightarrow P \mid P \fatsemi Q \mid P \Box Q \mid P \sqcap Q \mid \\ a : A \rightarrow P_a \mid P \rhd Q \mid P \triangle Q \mid f(P) \mid P \setminus A \mid \\ \|_{Ap}\, P \mid P\ {}_A\|_B\ Q \mid P ||| Q \mid P \underset{A}{\|} Q \mid \mu\, X \bullet F(X) \end{cases}$$
$$a ::= x \mid c.a$$

donde $a, x \in \Sigma$; $A, B \subseteq \Sigma$; $P, Q \in \mathbf{P}$; y $c \in \mathbf{C}$.

2.2.3 Cálculos de procesos temporizados

Dentro del grupo de lenguajes derivados de CSP, con el objetivo de poder describir Sistemas de Tiempo Real (STR) y sistemas con criticidad con mayor precisión, cabe mencionar Procesos Secuenciales de Comunicación Temporizados (*Timed CSP*) [92, 184] y Procesos Secuenciales de Comunicación + Tiempo (*Communicating Sequential Processes + Time*, CSP+T) [208, 209], siendo el segundo una aproximación más simple que la primera pero lo suficientemente completo para describir formalmente un conjunto de procesos deterministas con restricciones temporales [209]. Dada la importancia que Timed CSP y CSP+T tienen para nuestro trabajo, en las próximas secciones se profundizará en estos lenguajes. Además, se justificará la selección de CSP+T para el desarrollo de este trabajo.

2.2.3.1 Procesos Secuenciales de Comunicación Temporizados

Timed CSP [59] extiende a CSP [92] con primitivas temporales. Según lo indica su nombre, CSP es una notación basada en comunicaciones entre procesos que se pueden abstraer como eventos, orientada a describir la secuenciación del comportamiento dentro de un proceso y la sincronización del comportamiento (o de la comunicación) entre los procesos. Timed CSP amplía CSP introduciendo la capacidad para considerar aspectos temporales de orden y de sincronización durante la ejecución de un proceso. A las primitivas de proceso estándares de CSP, Timed CSP agrega dos primitivas específicas de tiempo, *retardo* (*WAIT*) y *timeout* ($\triangleright$) [128]. El operador *WAIT* es una forma retardada de *SKIP*; no hace nada, sino que saldrá del proceso con éxito después del tiempo especificado.

Un proceso que no establece ninguna comunicación para el período t y después termina, se escribe *WAIT* t. El proceso *WAIT* $t \fatsemi P$, se utiliza para representar el retardo de P durante un tiempo t. Una expresión de la forma $a \rightarrow \mathit{WAIT}\ \delta;\ P$ es generalmente más usada que la expresión $a \xrightarrow{\delta} P$, especialmente en procesos complejos.

La construcción timeout ($\triangleright_t$) pasa el control a un manejador de excepción si no hay ocurrencia del evento especificado en el proceso primario antes de un cierto plazo [128]. El proceso $u \rightarrow P \triangleright_t Q$ pasará el control a Q si el evento a no ha ocurrido en el tiempo t, desde la invocación del proceso.

En resumen, el lenguaje Timed CSP es definido por las reglas gramaticales [59] de la definición 2.16.

Definición 2.12. Sintaxis de Timed CSP. Dado un conjunto Σ de símbolos, que pueden ser compuestos con un conjunto de canales de comunicación **C**

usando el operador punto (.), un conjunto $\mathbf{T} \in \mathbb{R}^+$ de variables de tiempo, y un conjunto de nombres de procesos **P**, la *sintaxis de Timed CSP* es dada por:

$$P, Q ::= \begin{cases} STOP \mid SKIP \mid WAIT\, t \mid a \rightarrow P \mid P \fatsemi Q \mid P \Box Q \mid P \sqcap Q \mid \\ a : A \rightarrow P_a \mid P \rhd_t Q \mid P \triangle Q \mid f(P) \mid P \setminus A \mid \\ \|_{Ap}\ P \mid P\ {}_A\|_B\ Q \mid P ||| Q \mid P \underset{A}{\|} Q \mid \mu\, X \bullet F(X) \end{cases}$$
$$a ::= x \mid c.a$$

donde $a, x \in \mathbf{\Sigma}$; $A, B \subseteq \mathbf{\Sigma}$; $P, Q \in \mathbf{P}$; $t \in \mathbf{T}$; y $c \in \mathbf{C}$.

El lenguaje y los modelos de Timed CSP han experimentado una evolución gradual desde el trabajo descrito en [175] hasta lo publicado en [58], permitiendo manejar los aspectos de retardo y timeout de manera simple y elegante [59]. Timed CSP captura las restricciones temporales que se aplican a una especificación formal de un sistema por la composición paralela de un conjunto de procesos, en donde en cada uno de los cuales se puede describir una restricción temporal específica. Estas restricciones de tiempo son vistas como refinamientos temporales del sistema, ya que se escribe un proceso más simple para representar la especificación de cada restricción de tiempo, lo que facilita la especificación y las pruebas temporales del sistema. Sin embargo, en algunos casos, esto es imposible sin extender la sintaxis de Timed CSP con operadores que permitan registrar y acceder al registro del tiempo en el cual se anotan eventos específicos que ocurren en la ejecución del sistema.

Por lo anterior, se describe a continuación el lenguaje de especificación CSP+T, indicando las ventajas que éste tiene sobre los anteriores para expresar de una forma más completa los aspectos temporales de sistemas con criticidad.

2.2.3.2 Extensión y operadores de Procesos Secuenciales de Comunicación + Tiempo

En 1991 surge el lenguaje Procesos Secuenciales de Comunicación + Tiempo (*Communicating Sequential Processes + Time*, CSP+T) [208, 209] como un lenguaje formal de especificación que agrega expresividad a ciertos aspectos secuenciales del CSP y permite la descripción de sincronizaciones complejas de eventos dentro del código de especificación de un proceso secuencial, proporcionando la posibilidad de describir restricciones temporales que afectan a la ocurrencia de eventos de la mayoría de los STR [209]. Este tipo de especificación, por ejemplo, no se puede hacer sólo utilizando los constructos de sincronización de Timed CSP como *WAIT* [209]. CSP+T

es un lenguaje de especificación formal adecuado para la mayoría de los STR [209]. Su sintaxis viene dada por la Definición 2.13.

Definición 2.13. Sintaxis de CSP+T. Dado un conjunto Σ de símbolos, que pueden ser compuestos con un conjunto de canales de comunicación **C** usando el operador punto (.), un conjunto **M** de variables marcadoras, un conjunto **I** de intervalos de tiempo, y un conjunto de nombres de procesos **P**, la *sintaxis de CSP+T* es dada por:

$$\begin{aligned} P, Q ::= {} & \mathit{STOP} \mid \mathit{SKIP} \mid t0.\star \rightarrow \widetilde{P} \mid a \bowtie v \rightarrow P \mid P \fatsemi Q \mid P \Box Q \mid \\ & P \sqcap Q \mid P \backslash A \mid P \triangle Q \mid P \| \| Q \mid P \underset{A}{\|} Q \mid \mu X \bullet P \mid \\ & \mathrm{I}(T, t_a).a \rightarrow P \mid \mathrm{I}(T, ta) \rightarrow \widetilde{P} \end{aligned}$$

$$\widetilde{P} ::= a \rightarrow P \ \text{(término de proceso guardado)}$$

donde $a, \star \in \Sigma$; $A \subseteq \Sigma$; $v \in \mathbf{M}$; $\mathrm{I} \in \mathbf{I}$; $P, Q, X, \widetilde{P} \in \mathbf{P}$; $c \in \mathbf{C}$; $\mathrm{I}(T, t_a) = \{t \mid rel(t_a, v) \leq t \leq rel(t_a + T, v)\}$ donde $rel(x, v) = x + v - t0$, $t0$ corresponde al evento de instanciación ($\star$) que lo precede, ocurrido en algún tiempo absoluto t_0 y el valor guardado en la variable v es x; es decir, $\mathrm{I}(T, t_a)$ es el intervalo $[ta, ta + T]$ de habilitación del evento a respecto a la variable marcadora v.

Las principales extensiones realizadas en CSP+T con respeto al CSP son [39, 21, 136, 208, 209]:

- Cada proceso P define su propio conjunto de símbolos de comunicación, el *alfabeto de comunicación* $\alpha(P)$ ($\alpha(P) \subseteq \Sigma$). Estas comunicaciones representan los eventos que el proceso P recibe de su entorno (constituido por todos los otros procesos del sistema) o que ocurren internamente. Los eventos externos pueden ser entendidos como sincronización entre un proceso asíncrono y su entorno. Los eventos internos, como el evento τ, no están visibles externamente. Cualquier tipo de evento causa un cambio de estado observable en el proceso.
- Las acciones de comunicación de un proceso P ($Comm\_act(P)$) contiene todas las formas de comunicación de CSP (?, !, sincronización, uno a uno, comunicación entre procesos paralelos) en las cuales el proceso P puede implicarse ($Interface(P)$), incluyendo el alfabeto de comunicación $\alpha(P)$ que representa las señales y eventos que ocurren en P. Por ende, las comunicaciones del proceso P están dadas por el conjunto $Comm\_act(P) = (Interface(P) \cup \alpha(P))$.
- Se introduce el operador $\star$ (estrella) en la notación para denotar la instanciación de los procesos secuenciales. Hay que crear una instancia de un término de procesos antes de que pueda ejecutarse, siendo este evento (el de instanciación) único en el sistema y representa el origen del tiempo

global del sistema en el cual los procesos pueden comenzar su ejecución. Como ejemplo, considérese un proceso que inicialmente sólo puede implicarse en la ocurrencia del evento a. En CSP este proceso se denota como: $P = a \rightarrow STOP$, pero en CSP+T este proceso debe instanciarse antes de ejecutarse. Sea P' la versión en CSP+T de P, donde 0 es el tiempo en el cual se instancia P' y t es el tiempo de ocurrencia del evento a, entonces la especificación de P' es: $P' = 0.\star \rightarrow t.a \rightarrow STOP$, donde $t \in (0,\infty)$.

- Se incorpora el operador $\bowtie$ junto con una variable marcadora para registrar el instante del tiempo en el cual ocurre un evento. Escribiendo $ev \bowtie v$, se indica que el tiempo en el cual ocurre ev durante la ejecución de un proceso se guarda en la variable v. Este tiempo viene dado por un número real positivo, de modo que una secuencia de eventos es monótona no decreciente. Otros eventos posteriores pueden instanciar la misma variable temporal con otros valores. En la especificación del proceso $P = 0.\star \rightarrow a \bowtie var \rightarrow$ ***STOP***, la variable var guardará el valor correspondiente al tiempo en que ha ocurrido el evento a y, por tanto, se cumplirá siempre que $var > 0$. Las variables asociadas al operador $\bowtie$ se llaman *variables marcadoras* y su ámbito está limitado a un proceso secuencial. No pueden ser referidas, asignadas o manipuladas de ninguna otra forma por procesos diferentes al que están asociadas en una composición paralela.
- Cada evento está asociado a un intervalo de tiempo que se llama *intervalo de tiempo de activación* del evento y es continuo. Este intervalo representa el período de tiempo durante el cual el evento se considera disponible para el proceso y su entorno, y su inicio es relativo a la ocurrencia de eventos anteriores al proceso actual en ejecución. Un proceso se considera como un proceso *STOP* si éste no se puede implicar con ningún evento alternativo después de que expire el intervalo de activación del evento. Por ejemplo, supóngase que existe cierto proceso P que solamente puede implicarse en el evento a, y a sólo puede tener lugar después de 1 unidad de tiempo y hasta 2 unidades de tiempo desde el momento en que se instanció (evento marcador) ese proceso P, registrándose, además, el tiempo en que ha ocurrido el evento a en la variable marcadora v. La especificación es: $P = 0.\star \rightarrow (1,2].a \bowtie v \rightarrow STOP$. Después de la ejecución del proceso, el valor de la variable marcadora cumplirá la desigualdad $1 < v \leq 2$.

  Cuando no existen variables marcadoras el intervalo de activación se define como relativo al evento inmediatamente precedente. Los intervalos de activación pueden estár también definidos en términos de funciones sobre el conjunto de variables marcadoras. Es decir, los intervalos de activación pueden denotarse de una manera más compacta usando la función

I(T, v), donde v es la variable marcadora que registró el instante de tiempo cuando ocurrió el evento precedente y T define la duración del intervalo de tiempo que comienza en el instante de tiempo guardado en v. Un ejemplo es: $P = 0.\star \rightarrow a \bowtie v \rightarrow \text{I}(3, v).c \rightarrow d \rightarrow \textit{STOP}$, donde el evento c debe ocurrir dentro de las tres unidades de tiempo siguientes al instante en el cual el proceso P se implique en el evento a. Si la variable marcadora no aparece en el argumento de la función I, quedando I(T), el intervalo de activación $(t, t + T]$ es relativo a la variable marcadora v ($v \equiv t$) previa en el ámbito del proceso. En caso de que no apararezca el citado intervalo de activación, el intervalo de activación para el proceso se considera el intervalo $(0, \infty)$. Los tiempos para los eventos son absolutos y los tiempos de comienzo de los intervalos son relativos al tiempo previo guardado en la variable marcadora que aparece en su definición.

- La semántica de la composición paralela de dos procesos ($P \parallel Q$) con intervalos de activación (E_1, E_2) que deben sincronizarse, depende de si los valores de estos intervalos son idénticos ($E_1 = E_2$), parcialmente superpuestos ($E_1 \cap E_2 \neq \varnothing$) o disjuntos ($E_1 \cap E_2 = \varnothing$). En el primer caso, los procesos se sincronizan con respecto a la ocurrencia del evento inicial (que es común), como en la semántica de comunicación de CSP; es decir, dados $P = E_1.Q$ y $R = E_2.S$, entonces $P \parallel Q \neq \textit{STOP}$ sí y sólo si $\alpha(Q) \cap \alpha(S) \neq \varnothing \wedge E_1 \cap E_2 \neq \varnothing$. En el caso de que los intervalos de activación sean disjuntos ($E_1 \cap E_2 = \varnothing$) la composición paralela de procesos se comporta como el proceso *STOP*.

Como se puede ver, CSP+T extiende el lenguaje CSP asignándole tiempos a la ocurrencia de los eventos, de dos maneras:

1. todos los eventos han de ocurrir dentro de un intervalo de tiempo de activación, sobre el cual se espera que los eventos ocurran una sola vez durante una ejecución particular, y
2. esos intervalos de tiempo pueden ser expresados en términos de un conjunto arbitrario de eventos marcadores dentro de la ejecución del proceso.

Para la semántica que proponemos, estos dos aspectos son súmamente importantes. Los retomaremos en el capítulo 3 cuando presentemos el análisis temporal de los elementos notaciones de tiempo de BPMN y cómo a través de CSP+T podemos precisar y controlar la oportunidad de ocurrencia de dichos elementos.

2.2.3.3 Refinamiento hacia el tiempo

El proceso de verificación de términos de procesos temporizados, tales como los escritos con Timed CSP o CSP+T, puede ser difícil, especialmente si

se requiere usar múltiples relojes de modelo continuo (o denso) de tiempo para describir cualquier comportamiento del modelo del sistema. Sin embargo, hay varios enfoques sobre la base de trazas y fallos temporizados [190, 224], que en determinadas condiciones permitan verificar procesos temporizados haciendo extracción de anotaciones temporales de las trazas y luego realizando una prueba de verificación de refinamiento entre términos sintácticos en un cálculo de procesos no temporizados. En este caso, las demostraciones de refinamiento son condiciones necesarias para demostrar que las propiedades temporales son satisfechas por un modelo del sistema. Son un importante pre–requisito para lograr un comportamiento correcto de un sistema temporizado, principalmente para caracterizar el comportamiento reactivo correcto de dichos sistemas.

Dado que el EFVC que se define en el capítulo 4, objeto principal de esta tesis doctoral, utiliza términos de proceso CSP+T para especificar los modelos del sistema y de las propiedades, y para facilitar la comprobación de las mismas, llevamos a cabo una adaptación de la teoría de refinamiento hacia tiempo (*timewise refinement*) [190, 224] entre el modelo no temporizado CSP y el modelo temporizado CSP+T. Nos concentramos en dos relaciones, ambas desde modelos no temporizados a modelos temporizados:

- una desde el modelo de trazas no temporizado, utilizado para el análisis de aspectos de seguridad, y
- tra desde el modelo de trazas infinitas no temporizadas, utilizado para el análisis de aspectos de vivacidad, que contiene fallos.

Estas relaciones son formalmente definidas en las Definiciones 2.14 y 2.15 a continuación.

Definición 2.14. Refinamiento hacia trazas temporizadas. Para un proceso no temporizado CSP P y un proceso temporizado CSP+T Q, la relación de refinaminto hacia trazas temporizadas $P \sqsubseteq_t Q$, sostiene:

$$P \sqsubseteq_t Q \;\Leftrightarrow\; strip(s) \in \mathsf{traces}(P) \tag{2.5}$$

es decir, dada una traza finita temporizada s de Q, la misma traza con las variables marcadoras y los intervalos de activación removidos —$strip(s)$— es una traza finita de P.

Definición 2.15. Refinamiento hacia fallos temporizados. Para un proceso no temporizado CSP P y un proceso temporizado CSP+T Q, la relación de refinaminto hacia fallos temporizadas $P \sqsubseteq_f Q$, sostiene:

$$P \sqsubseteq_f Q \;\Leftrightarrow\; (strip(s), ref) \in \mathbf{SF}[\![P]\!] \tag{2.6}$$

es decir, en cualquier momento que $(s, [t, \infty) \times ref)$ es un fallo temporizado de Q, entonces (strip(s),ref) es un fallo de P: eventuales rechazos ref de Q, siempre corresponden al conjunto de rechazos no temporizados ref.

Consideramos que una traza no temporizada es una descripción abstracta de un fallo temporizado, si la traza no temporizada corresponde a la secuencia de eventos en la traza temporizada, de acuerdo con las definiciones anteriores. En resumen, los modelos semánticos de tiempo de CSP y de CSP+T, en conjunto, permiten el análisis de los sistemas y de sus componentes en diferentes niveles de abstracción. En particular, las propiedades de seguridad y vivacidad no temporizadas de un modelo de sistema especificado con CSP+T pueden ser verificables en el modelo no temporizado CSP, y luego ser usado en un análisis temporal.

Las Definiciones 2.14 y 2.15 son de suma importancia en nuestro enfoque (ver capítulo 4), debido a que en la actualidad no se cuenta con herramientas de verificación automática (ver sección 2.4 más adelante) para cálculos de proceso temporizados derivados de CSP. Por ello, las verificaciones deben hacerse utilizando la herramienta FDR2 [71], la cual no maneja tiempo.

Vistas las lógicas temporales más conocidas a la fecha, así como los cálculos de procesos más adecuados para la especificación y verificación de sistemas, en la próxima sección trataremos los aspectos más resaltantes de la verificación formal.

2.3 Verificación

Verificar significa *salir cierto y verdadero lo que se dijo o pronosticó* [152]. En este sentido, la verificación de software es el proceso a través del cual se corrobora que el software satisface sus objetivos [172, 192, 196]; es decir, lo que debía ser. A continuación exponemos los conceptos que giran en torno a la verificación de software y de sistemas con criticidad, que definen el contexto del enfoque de verificación composicional que se propone en esta tesis doctoral para aplicar en el ámbito de los BP y que se describe en el capítulo 4.

2.3.1 Verificación y razonamiento deductivo

El razonamiento deductivo se mueve de lo general a lo particular. Toma una premisa general y deduce conclusiones particulares [12]. Un argumento de-

ductivo *válido* es aquel en el que la conclusión necesariamente se deriva de la premisa. Puede ser que la premisa no sea *verdadera* pero, no obstante, la forma del argumento es *válida*. Un argumento deductivo *válido* contendrá algo en la conclusión totalmente nuevo e independiente de aquellas cosas mencionadas en la premisa del argumento. En el razonamiento deductivo, la conclusión no contiene más información semántica que las premisas a partir de las que se ha obtenido. Además, la conclusión resulta de una simplificación de la información y ésta no repite información que se presenta explícitamente en alguna de las premisas [12].

El esquema deductivo clásico para la verificación de Sistemas de Transición Equitativos (*Fair Transition Systems*, FTS) está basado en reglas de verificación, las cuales reducen las propiedades temporales del sistema a premisas temporales simples o de primer orden [130]. La verificación deductiva de programas es un método general para establecer propiedades de programas [72, 126]. Se parte de una especificación formal que describe algún aspecto de lo que el software debe hacer (o evitar), y utilizando una herramienta específica basada en alguna lógica de programas, el ingeniero establece una propiedad o su ausencia. Este método incurre en un coste humano muy superior (en tiempo y entrenamiento). En contraste, el MC deductivo [194] busca integrar diversas herramientas de verificación, reduciendo así el esfuerzo por parte del ingeniero.

En síntesis, tomando en cuenta lo que indica [199] en cuanto a la deducción automática, el razonamiento deductivo puede ser aplicado a la verificación de sistemas, siguiendo una idea de resolución como la que se indica en la Defiición 2.16.

Definición 2.16. Resolución deductiva. Si la fórmula ϑ es una propiedad esperada del sistema y el axioma

$$\phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_n \Rightarrow \vartheta \tag{2.7}$$

está en el conjunto Γ de axiomas a cumplir por el sistema, entonces la *resolución deductiva* del problema se reduce a probar cada propiedad $\phi_1, \phi_2, \ldots, \phi_n$.

Según [199], la idea es que este es un procedimiento el cual define ϑ (lo principal) en términos de $\phi_1, \phi_2, \ldots, \phi_n$ (las partes), y la notación es invertida (algunas veces usando ":= " por $\Leftarrow$): $\vartheta \Leftarrow \phi_1, \phi_2, \ldots, \phi_n$.

2.3.2 Técnicas de verificación formal

Las técnicas de verificación formal más conocidas son [51, 192]:

Prueba de Teoremas (*Theorem Proving*, TP). Técnica en la cual las propiedades deseadas y el sistema están expresados como fórmulas en algún tipo de lógica matemática. Esta lógica es determinada por un sistema formal, el cual define un conjunto de axiomas y un conjunto de reglas de inferencia. La TP es el proceso de encontrar una prueba de la propiedad a partir de los axiomas del sistema.

Comprobación de Modelos (*Model Checking*, MC). Técnica basada en la construcción de un modelo finito del sistema para probar que una propiedad deseada se cumple en aquel modelo. La verificación se lleva a cabo como una búsqueda exhaustiva automática en el espacio de posibles estados del sistema cuya finalización está garantizada por lo finito del modelo. Se usan dos aproximaciones generales de comprobadores de modelos. En la primera, el comprobador de modelos temporal, las especificaciones son expresadas en una lógica temporal y los sistemas son modelados como sistemas de transición de estados finitos. En la segunda aproximación, las especificaciones están dadas como un autómata. La desventaja principal del MC es el problema de la *explosión de estados* (éste se tratará en la siguiente sección).

El MC es más fácil de usar que la TP porque es completamente automático y más rápido, tal como se explica en la siguiente sección.

2.4 Verificación automática

El área de la computación que estudia las técnicas matemático–computacionales que permiten verificar sistemas complejos se llama Verificación Automática, comúnmente conocida como*Model checking* (MC), ya que es la automatización de la comprobación de modelos [16], introducida en la sección anterior. Gracias a los comprobadores de modelos (*model checkers*) podemos estudiar situaciones de comportamiento *normales* y *anormales*; es posible experimentar de manera segura con errores que no se pueden estudiar en el entorno final de funcionamiento de los sistemas. Es allí donde la técnica de MC (*verificar el modelo producto del diseño y no el programa real*) ha ayudado en la verificación de sistemas complejos [81, 106] como pueden ser los BP con criticidad. Tres herramientas de verificación comúnmente usadas son SPIN [93], SMV [134] y FDR2 [71]. SPIN usa LTL para generar un comprobador de modelos de las especificaciones del sistema, mientras que SMV usa CTL. FDR2 se basa en la teoría de concurrencia fundamentada en CSP [92], ampliamente difundida por la capacidad de abstracción/refinamiento y composición de este lenguaje [197].

Una manera intuitiva de explicar cómo funcionan las técnicas de MC es [49]: primero, el sistema es modelado matemáticamente. Luego, el usuario especifica en alguna lógica temporal la propiedad que desea que el sistema satisfaga. Y, finalmente, utilizando técnicas automáticas de análisis, se verifica que la especificación sea cierta en el modelo del sistema. Como estas técnicas son automáticas y además basadas en propiedades matemáticas, podemos estar 100% seguros de la certeza de la respuesta del procedimiento. Por supuesto, siempre al modelar el sistema podemos estar pasando por alto algo de su comportamiento. Es por eso que las técnicas de MC utilizan también técnicas de refinamiento [92, 185, 190], que permiten la verificación de modelos cada vez más parecidos al sistema original [16, 20].

El MC permite comprobar si un determinado programa cumple un conjunto de propiedades deseadas, como invariantes, la ausencia de interbloqueos o posposición indefinida, de forma automática. En esta técnica se analizan todos los posibles estados del programa para demostrar (o negar) que éste satisface una determina propiedad. Las propiedades suelen especificarse, en la mayoría de los casos, usando lógicas temporales como LTL o CTL, o en alguna lógica temporal derivada de éstas como CCTL [118, 187], TCTL [97] o MLTL [161], entre otras. El MC es, hoy por hoy, no un conjunto de técnicas teóricas sin trascendencia en la práctica, sino más bien un paradigma de cómo una teoría computacional puede ser transferida hacia productos industriales con impacto tecnológico [16].

Para proceder con la verificación mediante una herramienta de MC, tomamos la propiedad ϕ, representada por el autómata $\mathbf{M}$, originalmente expresada a través de una lógica temporal (véase la sección 2.1), $\neg\delta$ (usado para denotar que no existen bloqueos), también representado por un autómata, y el comportamiento del componente C, representado por el autómata $\mathbf{C}$, derivado del modelo del sistema desarrollado en un lenguaje formal (véase la sección 2.2). Entonces, la alcanzabilidad ($alcan()$) del estado $fallo$ se comprueba para demostrar que:

$$\mathbf{C} \,\|\, \mathbf{A} \,\|\, \mathbf{C}^T \vDash \neg alcan(fallo) \;\Rightarrow\; \mathbf{C} \vDash (\phi \wedge \neg\delta)\,,$$

donde, alternativamente, un autómata de ensayo $\mathbf{C}^T$ se puede construir, que incluye un estado $alcan(fallo)$ que es alcanzable si y sólo si ϕ y $\neg\delta$ no se cumplen. Si la verificación de un componente no es exitosa, la herramienta de MC, en general, presenta un contraejemplo para corregir el error en el modelo del componente o en la especificación de propiedad que se comprueba.

El MC fue inicialmente creado para sistemas finitos y una lógica temporal particular, CTL* [49]. Actualmente aplica a un escenario mucho más general y se han desarrollado varios algoritmos para otras lógicas temporales

(por ejemplo, ver [76, 96, 137]). Los algoritmos originales de MC trabajaban de forma explícita con todas las transiciones y estados de las EK (ver Definición 2.10) que representan tanto a las propiedades como el modelo del sistema. Pero éstos se han visto afectados por el problema de la *explosión de estados*. Por ello, una gran parte de la investigación en el área del MC intenta buscar métodos y técnicas que resuelvan parcialmente este problema. A continuación ahondamos un poco en las implicaciones de este problema.

2.4.1 Problema de la explosión de estados

La demostración de la ausencia de errores en la verificación exhaustiva de sistemas con criticidad no es algo sencillo. Esto debido principalmente al problema de la *explosión combinatoria de estados* cuando muchos subsistemas/componentes dependen unos de otros. El problema de la explosión de estados emerge debido a que la verificación requiere demostrar/comprobar la corrección de un gran conjunto de componentes, pudiendo llegar a alcanzar los límites de tiempo y espacio [31, 49, 81, 106] de cualquier comprobador de modelo, no estando, por tanto, excluida la aparición de fallos debido a dichas restricciones.

Por la naturaleza de los sistemas con criticidad, las partes críticas de éstos son representados como sistemas con un número finito de estados y verificados, como se vió en la sección anterior, utilizando MC [49]. Sin embargo, el problema principal del MC es la explosión de estados, ya que el tamaño del modelo del sistema puede llegar a ser muy grande y, en consecuencia, no manejable completamente y de manera explícita por los recursos computacionales del ordenador.

A continuación se describen brevemente algunas de las técnicas que se aplican en el MC para solventar este problema.

2.4.2 Técnicas

Para mitigar el problema de la explosión de estados se han propuesto diferentes técnicas como la *simetría* (explotar al máximo la estructura simétrica del sistema), la *representación simbólica* (la representación simbólica de un conjunto de estados) [16, 49], la *verificación composicional* [16, 106, 182, 226] el *uso de órdenes parciales* [16, 49] la *sustitución de comunicación asíncrona por comunicación síncrona* [16, 49, 182], cuando es posible, entre otros [111].

2.4.2.1 Simbólica

Los primeros algoritmos de MC usaron una representación explícita de EK como un grafo dirigido etiquetado [45, 47, 173]. Un gran avance se logró con el uso de las representaciones simbólicas [32, 34, 43, 54, 118, 132, 167, 187, 203], basadas en el uso de *Ordered Binary Decision Diagrams* [28, 29] (BDDs, para abreviar). Los BDDs son una representación de fórmulas booleanas, las cuales son canónicas una vez que ha sido establecido el orden de las variables. Intuitivamente, un estado del sistema es representado simbólicamente por una asignación de valores booleanos para el conjunto de variables de estado. Una *fórmula booleana* (y por tanto, su BDD) es una representación compacta de la serie de estados representados por las asignaciones que hacen verdadera a la fórmula [28, 29]. Del mismo modo, la relación de transición puede ser expresada como una fórmula booleana en dos conjuntos de variables, una en relación con el estado actual y el otro relacionado con el siguiente estado.

El uso de BDDs permite verificar sistemas muy grandes (más de 10^{20} estados) [34, 33, 132]. El MC simbólico ha tenido éxito en diversos campos, lo que permite el descubrimiento de errores de diseño que son muy difíciles de detectar con las técnicas tradicionales. Una extensa discusión de las aplicaciones del MC simbólico puede encontrarse en [51].

2.4.2.2 Acotada

La técnica denominada Verificación Automática Acotada (*Bounded Model Checking*, BMC) fue propuesta en [26]. No resuelve el problema de la complejidad de la verificación del modelo, ya que todavía se basa en un procedimiento exponencial y por lo tanto, su capacidad está limitada. Según [44] los experimentos han demostrado que pueden resolver muchos casos que no pueden ser resueltos por las técnicas basadas en BDDs. BMC también tiene la desventaja de no poder probar la ausencia de errores en los casos más reales [44]. Por lo tanto, BMC viene a ser una herramienta de verificación automática, pero que no sustituye a ninguna de ellas [16].

La idea básica del BMC es buscar un contraejemplo de las ejecuciones cuya longitud está limitada por algún entero k. Si no se encuentra algún error entonces k aumenta hasta que se encuentra un error o que el problema se vuelva inmanejable, o el límite superior conocido es alcanzado (este límite se denomina el *umbral de completitud del diseño* [44]). Problemas de satisfacción de proposiciones con BMC pueden ser resueltos por métodos denominados Problema de Satisfacción Booleana (*Boolean SATisfiability Problem*, SAT) en lugar de BDDs. Procedimientos SAT no sufren

el problema de explosión del espacio de estados de métodos basados en la BDD [44], facilitando que herramientas SAT pueden manejar los problemas de satisfacibilidad proposicional con cientos de miles de variables o más [44].

2.4.2.3 Abstracta

Otra técnica para atacar el problema de explosión de estados es la *interpretación abstracta* [55, 56, 105, 189]. El MC abstracto [48, 57, 124] combina la interpretación abstracta [56] y el MC [47, 49] para mejorar la verificación automática de grandes sistemas [10]. La aplicación del MC abstracto consiste en la abstracción tanto del modelo a analizar M como de las propiedades que deben verificarse en el modelo. El modelo abstracto M^+ es una sobre–aproximación del modelo concreto M, lo que significa que cada traza de ejecución concreta posible es imitada en el modelo abstracto. Este enfoque permite la verificación de propiedades que se refieren a todos los posibles caminos de comportamiento. Dos técnicas han sido desarrolladas con éxito para la construcción de M^+. El enfoque de la *abstracción de predicados* consiste en sustituir algunas expresiones del modelo seleccionado con variables booleanas que permite simplificaciones importantes (por ejemplo, las utilizadas en la herramienta SLAM [17, 18]). El método de *abstracción de datos* reduce el tipo de determinados datos por la transformación de su dominio concreto original en un dominio aproximado y simple. Este segundo enfoque se ha aplicado para abstraer modelos en las herramientas Bandera [88] y αSPIN [75].

2.4.2.4 Reducción de orden parcial

Una manera de generar directamente espacios de estados reducidos es mediante la Reducción de Orden Parcial (*Partial Order Reduction*, POR) [15, 171, 79], sólo que no utiliza abstracción. La POR es un método para reducir el número de estados del espacio de búsqueda de los algoritmos de MC [79]. Para el caso de autómatas temporizados, la semántica de orden parcial ha provisto el fundamento para algoritmos y estructuras de datos eficientes en la reducción de estados [125]. Este método aprovecha el hecho que en modelos con alto grado de concurrencia, existen normalmente transiciones que, independientemente del orden relativo en que sean ejecutadas, conducen al mismo resultado [79].

La técnica del *conjunto persistente/aferrado* [79, 204] calcula un subconjunto demostrable suficiente del conjunto de transiciones habilitadas en

cada estado visitado, tal que está garantizado que transiciones habilitadas no seleccionadas no interfieran con la ejecución de los que han sido seleccionadas. El subconjunto seleccionado se llama un conjunto persistente [79, 204]. En contraste, la técnica del *conjunto inactivo* [79] explota la información exclusivamente sobre las dependencias entre las transiciones habilitadas en el estado actual, así como la información registrada sobre el pasado de la búsqueda. Ambas técnicas pueden ser utilizadas de forma simultánea y son complementarias [79].

2.4.2.5 Simétrica

Cualquier técnica de búsqueda utilizada en el MC consiste en la exploración del espacio de estados asociados con el modelo. La simetría inherente al sistema original se refleja en el espacio de estados. Por lo tanto, el conocimiento de la simetría del sistema se puede utilizar para evitar la búsqueda de las áreas de espacio de estado que son simétricamente equivalente a las áreas que hayan sido registrados previamente [65, 146]. De esta manera se reduce el espacio de estados que deben ser explorados automáticamente. Distintos enfoques y técnicas han sido propuestos para el uso de la simetría en el MC. Algunos de estos han sido aplicados dentro de comprobadores de modelos ampliamente utilizados como SPIN (SymmSpin [27]), UPPAAL [91], Murφ Verification System [100], SMV [132] y `red` [210]. Inclusive, existe un comprobador de modelo diseñado principalmente para la verificación de sistemas altamente simétricos, el *Symmetry–based Model Checker* (SMC) [195].

2.4.2.6 Herramientas

Ya a lo largo de este capítulo se han nombrado algunas herramientas de MC, las cuales se han destacado por su aplicación a la solución de problemas industriales; sin embargo, en este apartado resaltamos aquellas que se consideran con mayor difusión y disponibilidad de uso. En la Tabla 2.2 se listan algunas (no todas) herramientas de MC de dominio público, muchas de los cuales están disponibles gratuitamente para uso no comercial y pueden ser utilizadas comercialmente por el pago de licencia (en sus sitios web están detalladas las condiciones).

En la Base de datos de Herramientas de Verificación (*Verification Tools Database*, YAHODA[3], mantenida por el laboratorio *Parallel & Distributed Systems Laboratory* (ParaDiSe) de la Facultad de Informática de la Univer-

[3] `http://fi.muni.cz/yahoda/`

Herramienta	URL	Lenguaje de modelado	Lenguaje de especificación
SMV	http://www.cs.cmu.edu/~modelcheck/smv.html	Red de autómatas que se comunican mediante variables compartidas	CTL
SPIN	http://netlib.bell-labs.com/netlib/spin/	PROMELA	PLTL
KRONOS	http://www-verimag.imag.fr/TEMPORISE/kronos	Autómatas temporizados	TCTL
UPPAAL	http://www.uppaal.com/	Autómatas temporizados	TCTL
HyTech	http://embedded.eecs.berkeley.edu/research/hytech/	Autómatas híbridos lineales	TCTL o ICTL
Design/CPN	http://www.daimi.au.dk/designCPN	Redes de Petri Coloreadas	CTL
FDR2	http://www.fsel.com/	CSP	CSP

Tabla 2.2 Algunas herramientas de MC de dominio público.

sidad de Masaryk, está disponible una tabla comparativa con aproximadamente 50 herramientas de MC. Para la comparación se especifican las características principales de cada herramienta, así como la información completa de las herramientas listadas. La ventaja de la información que se suministra en YAHODA radica en que los datos de cada herramienta es incluida después de que ParaDiSe ha realizado una exhaustiva evaluación de ésta.

2.5 Verificación composicional

Una de las estrategias que han sido aplicadas para la generación de espacios de estados más pequeños y equivalentes con respecto a las propiedades deseadas del sistema (p.e., interbloqueo y accesibilidad), ha sido la conocida *verificación composicional*. La idea ha sido eliminar información irrelevante de la verificación para que ésta sea más eficiente. Esto además permite hacer el análisis de los modelos del sistema más factible según los recursos computacionales disponibles [81].

En la verificación composicional, la meta es comprobar las propiedades de los componentes de un sistema y deducir las propiedades globales del sistema a partir de esas propiedades locales. La especificación de un sistema es descompuesta en las propiedades de sus componentes las cuales son verificadas por separado. Si deducimos que el sistema satisface cada propiedad local, y mostramos que de la conjunción de las propiedades locales implica la especificación general, entonces podemos concluir que el sistema también satisface esta especificación.

Las teorías actuales de verificación composicional han sido desarrolladas principalmente para *sistemas basados en eventos*. En este sentido, las técnicas de verificación composicional *ayudan a analizar la capacidad de reacción de los estados del sistema, permitiendo así información útil para su entendimiento* [16, 182, 226]. Además, aquí es donde las herramientas de MC constituyen un gran soporte para reproducir con exactitud la secuencia de eventos (trazas) que se esperan del sistema con criticidad y que fueron especificados previamente en el lenguaje formal [16, 22]. Según [16, 111], lo más importante es que la propiedad de reacción de los estados puede eliminar la dificultad de corrección de errores y modificación de diseños incorrectos usando la verificación composicional, dado el hecho que la mayoría de las ocurrencias de los eventos son descartadas por las técnicas de verificación composicional con la finalidad de proveer un espacio pequeño de estados para el análisis. Sólo se trabaja con los estados del sistema y los eventos del entorno relevantes para la verificación, abstrayendo al proceso de verificación de aquellas ocurencias de eventos y alcance de estados irrelevantes para la comprobación de la propiedad deseada para el componente y el sistema. Esto contribuye a la minimización del problema de la explosión de estados.

2.5.1 Razonamiento composicional

El razonamiento composicional busca *hacer verificaciones locales y garantizar que al cumplirse las especificaciones parciales, la composición de éstas asegura que se cumplen las propiedades globales que se esperan del sistema* [16, 31, 81, 106, 182, 226]. El problema fundamental de la verificación composicional es probar que un sistema compuesto satisface su especificación si todos sus componentes satisfacen sus especificaciones [16, 49, 121, 133]. En general, la verificación composicional puede ser explotada efectivamente cuando el modelo del sistema es el resultado natural de una *composición*; en este sentido, un modelo que en sí mismo consiste de *módulos independientes* es adecuado para la verificación composicional [183].

El enfoque del razonamiento composicional reduce la verificación de una propiedad ϕ de un sistema $S(C_1, \ldots, C_n)$, ensamblado a partir de los componentes $C_1, \ldots, C_n$, a la verificación de las propiedades $\phi_1, \ldots, \phi_n$ de los componentes. Esto conlleva a tener en cuenta dos parámetros [174]:

1. el lenguaje de especificación L_{spec} en el que se formulan las propiedades, y

2. la colección de operaciones OP por las cuales un sistema complejo puede ser ensamblado a partir de sus componentes.

Lo que se busca de la verificación composicional (ver Teorema 2.1) es conseguir un algoritmo el cual para cada formula $\phi \in L_{spec}$ y cada operador n-ario $S \in OP$ pueda construir fórmulas $\phi_1, \ldots, \phi_n$ de tal manera que $S(C_1, \ldots, C_n)$ satisface ϕ si y sólo si C_1 satisface ϕ_1, C_2 satisface ϕ_2, ..., y C_n satisface ϕ_n.

Según [174], el *Teorema de Composición* (ver Teorema 2.1) se cumple cuando el lenguaje de especificación L_{spec} es una lógica multimodal y el conjunto de operaciones OP consisten en una gran variedad de operadores de producto ("composición paralela") generalizado. Por el contrario, si L_{spec} puede expresar *existe un camino tal que para todos los nodos del camino se cumple una propiedad p* o *existe en el futuro un nodo donde se cumple una propiedad p*, escritas normalmente como $\mathsf{EG}p$ o $\mathsf{EF}p$, respectivamente, el teorema de composición no se cumple para cada operador paralelo simple [174].

Según [182] podemos establecer que el teorema de composición se define como sigue:

Teorema 2.1. ***Teorema de composición.*** *Por cada operador $op^P \in OP'$ del lenguaje de programación del sistema debe existir un operador $op^S \in OP$ en el lenguaje de especificación L_{spec} de manera que:*

1. *Siempre que los programas P_i satisfagan las especificaciones ϕ_i para $i = 1, \ldots, n$, también tenemos para cada operador n-ario $op^P \in OP$ que $op^P(P_1, \ldots, P_n)$ satisface a $op^S(\phi_1, \ldots, \phi_n)$;*
2. *Siempre que $op^P(P_1, \ldots, P_n)$ satisface ϕ, existen especificaciones ϕ_i para cada programa P_i que satisface ϕ_i para $i = 1, \ldots, n$ y $op^S(\phi_1, \ldots, \phi_n) \Rightarrow \phi$ es válido.*

El teorema de composición implica las propiedades en él indicadas, pero éste no se deriva de ellas. Generalmente se pone de manifiesto que dos programas P_1 y P_2 satisfacen las mismas especificaciones en L_{spec}, pero puede haber un contexto $op[, Q]$ y una fórmula ϕ tal que $op(P_1, Q)$ satisface ϕ, mientras que $op(P_2, Q)$ no satisface ϕ. En tales situaciones, a fin de obtener un sistema de prueba composicional, L_{spec} debe ser sustituido por un lenguaje más expresivo. Los resultados muestran que el aumento en el poder expresivo de L_{spec} no es suficiente ayuda para obtener el teorema de composición [174].

2.5.2 Composicionalidad – composición paralela

En el campo de los métodos formales, la verificación composicional fue introducida por Hoare [92] y Miller [147] gracias a la capacidad del *operador de composición paralela* ($\|$) de CSP y CCS, respectivamente, para especificar la concurrencia de procesos [184], y ha sido extensamente explorado por los investigadores del área de los cálculos de proceso [111]. Por otro lado, ha tenido una gran aceptación para la especificación de comunicaciones síncronas, como forma natural de la interacción entre procesos. Además, a través del *operador de ocultamiento* ($\backslash$), explota los conceptos de abstracción/refinamiento, los cuales forman parte de la verificación composicional.

2.5.3 Razonamiento Asumir/Garantizar

La verificación composicional se soporta en el *Razonamiento Asumir/Garantizar* (RAG) [16, 49, 73]: *cada componente garantiza ciertas propiedades sobre la base de los supuestos de los otros componentes*. Por ejemplo, los componentes C_1 y C_2 pueden satisfacer su especificación $\mathbf{S}_1$ y $\mathbf{S}_2$, respectivamente, sólo si el entorno de cada componente se comporta como se espera que se comporten los otros componentes del sistema. De manera simple, si tenemos los componetes C_1 y C_2 de un sistema que trabajan en concurrencia, dado que el comportamiento de C_1 depende del comportamiento de C_2 y el comporamiento de C_2 depende del comportamiento de C_1, la verificación requiere especificar el conjunto de suposiciones que debe satisfacer C_2 y C_1 para garantizar la correctitud de C_1 y C_2, respectivamente. Por la combinación del conjunto de propiedades supuestas y garantizadas de C_1 y C_2 de una forma apropiada, es posible establecer la correctitud del sistema completo sin necesidad de construir el grafo completo de transición de estados del sistema.

Bajo el RAG, la expresión $\langle A\rangle\ M\ \langle P\rangle$, donde M es un componente, P es una propiedad y A es una suposición sobre el ambiente de M, es cierta si para cualquier M que sea parte de un sistema que satisface A, entonces el sistema debe también garantizar P. Para el caso de dos componentes C_1 y C_2, tenemos que la regla de prueba composicional simple muestra que si $\langle A\rangle\ C_1\ \langle P\rangle$ y $\langle true\rangle\ C_2\ \langle A\rangle$ se mantiene, entonces $\langle true\rangle\ C_1 \| C_2\ \langle P\rangle$ es verdad. Según [52], esta estrategia de prueba puede ser también expresada como una regla de inferencia como sigue:

$$\frac{\text{(Paso 1) } \langle A \rangle\ C_1\ \langle P \rangle \quad \text{(Paso 2) } \langle true \rangle\ C_2\ \langle A \rangle}{\langle true \rangle\ C_1 \| C_2\ \langle P \rangle}$$

Esto permite asegurar que las verificaciones locales realizadas a cada uno de los componentes del sistema, así como a la composición paralela de estos componentes, satisfacen las suposiciones y propiedades definidas para cada uno de ellos.

2.5.4 Enfoque modular – modularización

La verificación composicional explota el enfoque modular usado en el diseño de grandes sistemas. Con la modularización se reducen los costos de desarrollo de software porque permite el diseño independiente y en paralelo, sobre la base de un presupuesto disponible y un cronograma establecido, además que fomenta la mantenibilidad y la reutilización de software [35, 172, 196]. Estas ventajas son también parte del diseño de sistemas de software basado en componentes, razón por la cual este tipo de diseño tiene tanto auge en la actualidad [16, 182].

El proceso de modularización consiste en hacer una abstracción del sistema del cual se tiene inicialmente un panorama general. Se procede a *dividir* el sistema en partes pequeñas y simples [35, 172, 196]. Se parte de un primer módulo enunciando el sistema en términos de la solución que aporta al problema que lo motiva. Cuando un sistema es complejo o extenso, la solución consta de varios módulos que ejecutan tareas especificas. Así, cada uno de estos módulos es responsable de ejecutar una determinada tarea, lo que redunda en una mayor concentración, entendimiento y capacidad de solución a la hora de diseñar la lógica de cada uno de los módulos [35, 172, 196]. Dichos módulos son segmentos del algoritmo global del sistema. Algunos son módulos de control, encargados de distribuir el trabajo de los demás módulos. Estos módulos normalmente son considerados críticos, ya que ellos son los que gobiernan el funcionamiento del sistema [35, 123]. De esta manera se puede diseñar un esquema que indique la estructura general del algoritmo global de un sistema. Bajo el marco conceptual de UML [157], el concepto de módulo es conocido como *sub–sistema*, siendo los conceptos de modularización igualmente aplicables.

2.5.5 *Métodos formales para la verificación composicional de sistemas críticos*

La verificación de sistemas con criticidad es mucho más compleja que la análoga para los sistemas que no son críticos, debido a que su *correctitud* depende tanto de las restricciones de tiempo impuestas para su funcionamiento como de su propia funcionalidad. Una manera de probar la correctitud de sistemas con criticidad es usando *métodos formales*, ya que *ayudan a encontrar inconsistencias, ambigüedades e incompletitudes*. Es por ello que para este tipo de sistemas, la verificación formal es prácticamente mandatoria. Cuando hablamos de verificación formal, la misma consta de tres etapas [49]:

1. *Modelado.* Se construye un modelo matemático de los posibles comportamientos del sistema;
2. *Especificación.* Se especifica en un lenguaje formal los comportamientos (propiedades) esperados del sistema; y
3. *Verificación.* Se chequea si el modelo satisface la especificación.

Además, hablamos de que es formal porque el modelo y la especificación son objetos matemáticos y el análisis efectuado en el proceso de verificación responde con certeza si la especificación se cumple o no. Todo esto lleva a que el diseño de sistemas con criticidad deba estar soportado por un lenguaje formal que permita hacer una especificación de requisitos precisa y que pueda ser utilizado posteriormente en el proceso de verificación del sistema. Si se utiliza un lenguaje formal para la especificación de los requisitos del software, los diseños de alto nivel deben utilizar la misma notación, para así poder demostrar si los diseños cumplen dicha especificación; es decir, verificar (*¿se está fabricando correctamente?*) el sistema con criticidad.

Puesto que las actividades en el desarrollo de sistemas con criticidad pueden realizarse de manera aislada, se requiere de una notación que permita documentar cada etapa, de tal manera que se pueda lograr que las transformaciones de una etapa a otra sólo signifiquen realizar (en lo posible, automáticamente) traducciones o transformaciones de una notación a otra [35]. Para poder lograr esto es necesario que la notación tenga *base lógico–matemática*, tal como la que poseen algunos métodos formales (autómatas formales, cálculos de proceso, etc.). Mediante los lenguajes formales es posible hacer descripciones precisas; bien sea de la especificación de los requisitos o del diseño del sistema. Además, gracias a una especificación de los requisitos del sistema utilizando una notación formal, a través de actividades de verificación es posible probar que el diseño (tanto el de alto nivel como el detallado) satisface la especificación de dichos requisitos [35, 123]. Dada la necesidad de satisfacer los requisitos de alta fiabilidad de los sistemas

con criticidad, los diseñadores emplean, cada vez más, técnicas rigurosas de verificación.

Varias publicaciones sobre verificación de sistemas con criticidad pueden ser vistos en [168] y en `http://www.comlab.ox.ac.uk/archive/safety.html`. No obstante, podemos destacar los siguientes trabajos que consideramos más relacionados con los objetivos de nuestro trabajo. En orden cronológico, algunas iniciativas independientes para solventar la combinación de la explosión de estados, el uso de cálculos de proceso y la verificación composicional, relacionadas con la verificación de sistemas con criticidad, son brevemente referenciados a continuación.

En [50] se presenta un framework general que utiliza un proceso de interfaz adicional para modelar los procesos del entorno de un componente para reducir el crecimiento de los estados en el momento de la verificación. Según los autores, estos procesos de interfaz son normalmente mucho más simples que el entorno completo del componente (teorema de interfaz [50]). Esta técnica es más adecuada para sistemas débilmente acoplados. Cuando esta relación no se sostiene, el proceso de interfaz puede ser largo y no obtendremos una significativa reducción de estados para usar el teorema de interfaz.

En [222] toman ventaja de los operadores de las álgebras de procesos para controlar la explosión de estados y para permitir la verificación del comportamiento del sistema con respecto a sus propiedades. Según [222], los cálculos de proceso son composicionales en virtud de las leyes del álgebra de conjuntos. La estructura algebraica puede ser usada para elaborar una composición técnica para el análisis de alcanzabilidad. Por otra parte, [42] describe una técnica eficaz para aumentar la composición técnica para el análisis de alcanzabilidad con respecto a las limitaciones especificadas como contexto de los procesos de interfaz. Según los autores, el método propuesto es efectivo para reducir la barrera del cálculo por la aplicación de la verificación formal de sistemas concurrentes complejos [42]. Sin embargo, ambos trabajos no toman en cuenta aspectos de tiempo real [222].

En [81] se describe un framework para la verificación composicional de procesos de estados finitos. Identifican un subconjunto ∀CTL*, de CTL*, el cual es apropiado para el razonamiento composicional. De manera similar a [31] restringen el razonamiento a las propiedades globales comunes; es decir, las fórmulas temporales de CTL* que sólo contienen el cuantificador universal ∀. Este trabajo muestra cómo el RAG puede ser usado para la verificación composicional y disminuir la explosión de estados.

El método presentado en [106] verifica que las especificaciones a bajo nivel de abstracción implementen correctamente una especificación a alto nivel de abstracción, sin el apoyo de la técnica de MC y el uso de una lógica temporal. Especifican y verifican sistemas distribuidos que se comunican a

través de paso de mensajes asíncronos. Según los autores, el método es composicional; es decir, la especificación de un sistema compuesto puede ser obtenida de especificaciones de sus componentes. Así, la composicionalidad permite verificar que las especificaciones de un sistema compuesto se descomponen correctamente en las especificaciones de los componentes.

Por su parte, [31] afirma que su enfoque es composicional, aunque sea a costa de cierta generalidad. Restringen los requisitos de entrada a propiedades *globales* comunes; es decir, de la forma *en cada estado 'p' se mantiene*, *cada traza contiene un estado donde 'p' se mantiene*, *'p' se mantiene en todos los siguientes estados*. Logran la composicionalidad tomando un enfoque dependiente de la fórmula para reducir al mínimo cada modelo. Su algoritmo resulta esencialmente de una consulta de verificación en su dual, y progresivamente reduce cada modelo basado en la fórmula (negada).

En [112] se describe la aplicación de métodos formales para la verificación de un producto de software que implementa seguridad de e–commerce para Internet. La estrategia combina el MC con la verificación deductiva. El MC se utiliza para manejar los aspectos relacionados con el control de bloqueos, exclusión mutua y no interferencia. La verificación deductiva se utiliza para analizar las partes intensivas de datos del sistema. El lenguaje de especificación utilizado es LTL. Los autores sólo describen la verificación de propiedades de seguridad mediante MC y la metodología de verificación deductiva expuesta en [130].

Otro trabajo que aborda el problema de explosión de estados asociado al MC está en [52]. Se presenta un esquema basado en la verificación composicional, ejecutando el RAG de una manera gradual y completamente automático. Su enfoque itera un proceso basado en aprendizaje gradual de supuestos. Una de las ventajas del enfoque en [52] es su generalidad. Se basa en características estándares de comprobadores de modelos, y por lo tanto puede ser integrado con cualquier herramienta de MC.

La investigación de [78] aborda la verificación de sistemas complejos empotrados y distribuidos en tiempo real; en particular, para enfrentar el problema de explosión de estados para software de control de sistemas mecatrónicos, ofrececiendo un dominio específico para la definición semántica formal de un subconjunto del modelo de componente de UML 2.0 y una secuencia integrada de pasos de diseño.

Los trabajos en [73] y [126] se centran en la verificación formal de sistemas industriales en el ámbito de la ingeniería de procesos químicos que están guiados por Controladores Lógicos Programables (*Programmable Logic Controllers*, PLC), una clase de equipos automatizados de control. El RAG es usado para probar propiedades de módulos acoplados y para obtener conclusiones sobre el comportamiento de todo el sistema. Derivan

una especificación del comportamiento de un sistema de control industrial a partir de la descripción formal del comportamiento (individual) de sus componentes, por medio de un método de razonamiento composicional. Dado que las plantas químicas son un tipo de sistema con criticidad, este trabajo se puede extender para el diseño y la verificación de otros tipos de sistemas con criticidad.

Sobre la base del RAG, otra propuesta de MC composicional se encuentra en [52], que consiste en una infraestructura incremental que mantiene de forma automática la actualización de la prueba general de todo el sistema después de agregar nuevas construcciones a la fórmula de propiedad. La propuesta no es compatible con la verificación de las fórmulas de vivacidad. Además, sobre la base de RAG, en [177, 178] se describe un método para MC de los componentes con generación automática de asunciones. Este método genera un conjunto de reglas del entorno que representan el comportamiento de los eventos de la interfaz y se puede utilizar como asunciones del comportamiento de componentes. Según [177, 178], la nueva técnica es especialmente relevante para la detección de falsos positivos en la verificación con MC, ya que hay más asunciones tomadas en cuenta.

Otras propuestas basadas en RAG relacionados con CSP que vale la pena mencionar son [145], [211], y [149]. En [145], el foco está en la construcción de la descomposición del sistema cuando éste no está estructurado en componentes paralelos, o su estructura no es adecuada para un RAG eficiente. La técnica propuesta se basa principalmente en un conjunto de especificaciones formales en CSP–OZ; como consecuencia, se logra la modularización de las especificaciones. Los componentes en paralelo resultantes puede ser utilizados eficientemente para RAG.

Por su parte, en [149] se utiliza una relación de refinamiento de CSP para la presentación de una formulación simple de propiedades RAG en este cálculo de proceso, y una teoría que permite deducir las propiedades de sistemas compuestos a partir de la demostración de las propiedades de sus componentes. Según los autores, la teoría actual admite el razonamiento composicional sobre sistemas que pueden ser modelados como una composición paralela de dos procesos CSP. Entonces, los sistemas con múltiples componentes puede ser *razonados* de una forma jerárquica por la aplicación sucesiva de la teoría actual de los sistemas formados por dos procesos CSP. Los autores afirman que esto es relevante para lograr un *razonamiento efectivo de composición* usando el refinamiento de CSP.

Por último, en [211] se propone una técnica de composición para el control del refinamiento de trazas en CSP. La técnica utiliza un *algoritmo de aprendizaje de asunsiones* junto con el RAG básico. La demostración de propiedades de seguridad de procesos se lleva a cabo mediante la técnica *refinamiento de traza*, lo que permite la integración de la herramienta FDR2

[71] al proceso de verificación. Este trabajo nos inspiró en la forma de obtener la descripción de las propiedades CCLT como procesos CSP+T.

Algunas de las características de los trabajos citados anteriormente se muestran en la Tabla 2.3. En general, muchos autores han trabajado en la semántica de autómatas, ya que es el formalismo fundamental que subyace en las técnicas de MC. Sin embargo, los avances actuales en el uso del cálculo de procesos CSP son vistos como una posible estrategia alternativa para resolver el problema de la explosión de estados. CSP permite la obtención de un modelo del sistema que propicia la verificación composicional y el análisis del comportamiento de sus componentes, de acuerdo con los diferentes modelos semánticos (trazas, fallos, y divergencias), que son de aplicación a SCS y a BP, disminuyendo así el problema de la explosión de estados.

Resalta que todos los trabajos revisados se centran más en la búsqueda de la mejora de los algoritmos y técnicas de MC —como un medio de verificación a través de RAG— que en un uso integrado del MC con las técnicas de diseño de software y los métodos para la obtención de la verificación de un sistema completo o partes de él. Por ello, el trabajo realizado en [145] destaca, ya que hace hincapié en la necesidad de tener un modelo del sistema *composicional* para lograr la verificación de todo el sistema sobre la base del RAG. Esto indica que se requiere que el proceso de desarrollo de un software o de diseño de un BP se conciba bajo la visión de una estrategia de verificación desde el mismo momento que éste se inicia.

En resumen, a diferencia de otros trabajos sobre la verificación de la composición de SCS, el enfoque que proponemos a lo largo de esta tesis doctoral contribuye a la búsqueda de soluciones a la verificación de sistemas con criticidad, proporcionando una visión integral del desarrollo de estos sistemas mediante el uso de lenguajes formales que por naturaleza son composicionales (como los cálculos de proceso), y que establecen vínculos entre la especificación, el diseño, y la verificación de sistemas (que pueden ser de software o BP) de una manera natural.

Vistos los aspectos de la verificación composicional más relacionados con el trabajo que se expone a través de esta tesis doctoral, a continuación pasamos a establecer los tópicos referidos a los BP que enmarcan la propuesta que hacemos en próximos capítulos.

Formalismo	Trabajo	Estrategia	Propiedades	Herram.	Encapsulación
Autómatas	[50]	Modelado del ambiente usando procesos adicionales de interfaz.	Vivacidad, Exclusión mutua. Limitado a sistemas débilmente acoplados.	No	Si
	[81]	Uso de un orden previo con un subconjunto de la lógica CTL*.	Seguridad, Vivacidad simple y una noción de Equidad.	Ad-hoc	No
	[31]	Estima refinamientos incrementales usando modelos parciales.	Seguridad y Vivacidad.	Ad-hoc	No
	[112]	MC para tratar aspectos de control y método deductivo para manejar elementos del diseño intensivos en datos.	Seguridad.	SPIN	Si
	[52]	Realizan un RAG incremental.	Safety.	LTSA	Si
	[78]	Componen sistemas complejos de software a partir de patrones específicos al dominio.	Seguridad.	RAVEN	Si
	[73]	Análisis modular basado en el método RAG.	Seguridad.	HyTech	No
	[126]	Propone un enfoque de verificación modular.	Seguridad.	SMV	No
	[177, 178]	Generación automática de asunciones de componentes basado en el comportamiento del entorno.	No especifica.	SPIN	Si
Cálculo de procesos	[222]	Uso de la relación de equivalencia entre los procesos.	Interbloqueo, Falta de recursos y paralelismo dañino.	Ad-hoc	Si
	[149]	Una formulación simple de RAG usando CSP.	Safety.	FDR	Si
	[211]	Una técnica composicional para el chequeo de refinamiento de trazas.	Safety.	FDR	Si
	[145]	Construcción de descomposiciones para una RAG eficiente.	Seguridad.	FDR	Si

Tabla 2.3 Características principales de trabajos relacionados con la verificación de sistemas con criticidad.

2.6 Procesos de negocio

Actualmente las empresas organizan sus actividades por medio de un conjunto de BP [67, 120] para lograr los objetivos de su negocio. De acuerdo

con [67] los BP coordinan todas las acciones que realizan las organizaciones para agregarle valor a sus clientes, por lo que su correcta definición es vital para la supervivencia de éstas. En las próximas secciones expondremos los aspectos de los BP, en general, y de los BP con criticidad, en particular, lo que da un marco de referencia a los aportes que esta tesis doctoral brinda a los dominos del BPM y de la IS.

2.6.1 Definición — perspectivas.

Según [67], un BP es una colección de actividades diseñadas para producir una salida específica para un cliente o un mercado en particular. Esto implica un fuerte énfasis en *cómo* se realiza el trabajo dentro de una organización, en contraposición con un enfoque hacia el producto; en *qué* se produce.

Para [120], los BP se caracterizan por ser una colección de datos que son producidos y manipulados mediante un conjunto de tareas en las que ciertos agentes (tales como, trabajadores o departamentos) participan de acuerdo a un flujo de trabajo determinado. Además, estos procesos se hallan sujetos a un conjunto de reglas de negocio que determinan las políticas y la estructura de la información de la empresa. De allí que la finalidad del BPM es describir cada proceso del negocio, especificando sus datos, actividades (o tareas), roles (o agentes) y reglas de negocio [120]. Según [162], un BP es una serie estructurada de actividades específicas que no es realizada por una sola persona o departamento, sino que involucra a muchas personas/máquinas/sistemas de diferentes organizaciones que trabajan juntos para lograr un objetivo de negocio común. Por su parte, [69] dirige la definición de BP hacia el área informática, definiéndolos como un modelo del mismo, inteligible y procesable por aplicaciones software. Estos modelos se representan mediante lenguajes de definición de BP. Por otro lado, según [113], los procesos son secuencias especificas de actividades de trabajo a través del tiempo y del espacio, con un inicio, un final y unas entradas y salidas claramente definidas: *una estructura para la acción*. En este sentido, los BP tienen una vinculación con los *workflow*, tal como se expresa en [62] al indicar que éstos son la automatización de un BP, total o parcialmente, durante la cual documentos, información o tareas son pasadas de un participante a otro por una acción conforme a un conjunto de reglas procedimentales. En síntesis, consideramos para este trabajo que *los BP son una colección de actividades relacionadas lógicamente, que toman uno o más tipos de entradas y crean uno o más resultados que producen un valor para la organización, sus inversores o sus clientes, según unas reglas de negocio*.

Habitualmente, un BP está compuesto por la unión de aspectos de distintos ámbitos o perspectivas. Es común identificar las siguientes 5 perspectivas [67, 69]:

- *Control de flujo*. Define la coordinación de actividades; es decir, qué actividades deben ser ejecutadas y en qué orden. Se suelen utilizar primitivas de ejecución secuencial o en paralelo, bifurcación condicional, unión de ramas alternativas, iteración y sincronización, entre otras.
- *Recursos*. Define los actores (máquinas o humanos) necesarios para la ejecución del proceso. Los actores humanos, también llamados participantes, suelen estar agrupados según su rol o roles en la organización.
- *Datos*. Define la información de control y la de producción. Los datos de control se introducen para controlar la ejecución del proceso, mientras que los datos de producción se corresponden con las entidades del proceso.
- *Actividades*. Define las características de cada actividad del proceso (descripción, duración estimada, plazos, prioridad, clases de actores necesarias, etc.). Cada actividad puede estar compuesta por más de una acción elemental.
- *Operación*. Define las acciones elementales, que normalmente consisten en la ejecución de un sistema de software con unos determinados parámetros.

El trabajo que se desarrolla en esta tesis doctoral está relacionado directamente con las perspectivas de Control de flujo y Actividades. Nuestra propuesta permite definir con precisión y verificar restricciones temporales a nivel de actividades y del control del flujo de los BP, así como de la sincronización de actividades que intervienen en una colaboración entre BP.

2.6.2 Sistemas con criticidad — criticidad de procesos de negocio.

El término *Crtiticidad* es usado normalmente para cualificar la importancia de un equipo, actividad o proceso dentro de las operaciones de un negocio. Esta *importancia*, típicamente se basa en una evaluación de las consecuencias que implicaría un fallo del equipo, actividad o proceso, en el servicio que presta la organización. En este sentido, para este trabajo los *BP críticos* son *los BPs de una organización que son esenciales para cumplir la declaración de su misión*; es decir, aquellos que *deben ser restaurados inmediatamente después de una interrupción para garantizar la capacidad de la empresa afectada para proteger sus activos, hacer frente a sus necesidades fundamentales, y cumplir la normativa y los requisitos obligatorios*. A

menudo las necesidades de mejora de los BP críticos se debe a que éstos no llegan a brindar un rendimiento satisfactorio [135]; es decir, no propician de la mejor manera el logro positivo de los factores críticos de la organización. Como parte del rendimiento está el cumplimiento de plazos de tiempo que garanticen su correcta ejecución [135].

La criticidad está asociada con la *fiabilidad o confiabilidad* (en inglés, *reliability*) de aplicaciones software; es decir, con la probabilidad de que un equipo o sistema opere sin fallo por un determinado período de tiempo, bajo unas condiciones de operación previamente establecidas [110]. Adicionalmente, a nivel de los negocios, se habla de la *confiabilidad operacional*, referida a la capacidad de una instalación o sistema (integrado por procesos, tecnología y gente), para cumplir su función dentro de sus límites de tiempo y bajo un contexto de operaciones específico [200].

Con la automatización de BP y el soporte que las TIC le dan a los negocios, muchos sistemas se han convertido en el elemento fundamental para el funcionamiento de las empresas, así como para la realización de gran cantidad de procesos y actividades de la vida diaria. Como resulado de esto, algunos de los sistemas empresariales o aplicaciones software que utilizan las organizaciones se han catalogado como *sistemas críticos* (*de negocio*). Esto se debe a que su funcionamiento condiciona el trabajo diario de las organizaciones y de muchos aspectos de la vida humana, lo que incluye un buen tiempo de respuesta. Como consecuencia, estos sistemas necesitan ser verificados, certificados u homologados, antes de su puesta en funcionamiento [181].

Es necesario entonces invertir esfuerzo en la verificación de los BP que van a ser implementados a través de los sistemas, para minimizar posibles omisiones o inconsistencias con respecto a lo que se espera de ellos, tanto a nivel funcional como a nivel de rendimiento de acuerdo a restricciones temporales. Esta necesidad se ve incrementada cuando los BP de una organización forman parte de WS.

2.6.3 Servicios Web – coordinación y colaboración

El desarrollo de software de una aplicación web se caracteriza por la tendencia de generar paquetes modulares de software independientes de la plataforma, denominados Servicios Web (*Web Services*, WS). Su aporte principal es proveer funcionalidades automatizadas de BP a otras aplicaciones a través de Internet. Adicionalmente, permiten el desarrollo de tecnologías e investigaciones que los hacen adecuados para soluciones en dominios de tipo distribuido [201].

Bajo el enfoque WS los servicios desempeñan distintos roles dentro del BP de un negocio. La interacción y envío de información a través de los distintos roles o participantes en BP se conoce como *Coordinación de Servicios* [164]. Últimamente se emplean dos conceptos complementarios entre si que permiten diseñar una Coordinación de WS, estos son *Coreografía* y *Orquestación* [153]. Los cuales son confundidos con frecuencia debido a que ambos son participes del proceso de *Composición de WS*.

La *Orquestación* enfatiza en la descripción del proceso de composición desde un punto de vista individual, invocando mensajes y enviándolos, aprovechando la interfaz que provee la descripción del servicio [99]. Por su parte, la *Coreografía* presenta la definición de una colaboración entre servicios para alcanzar un objetivo en común, basada en el intercambio público de mensajes que ocurre entre estos [166]. Las diferencias entre orquestación y coreografía están basadas en analogías: la orquestación describe un control central del comportamiento como un director en una orquesta, mientras que la coreografía trata sobre el control distribuido del comportamiento donde participantes individuales realizan procesos basados en eventos externos, como en una danza coreográfica donde los bailarines reaccionan a los comportamientos de sus pares.

El intercambio de mensajes es clave en la definición de la coordinación de WS [153, 164]. En este sentido, la coreografía de WS debe ser descrita desde un punto de vista tanto estructural como dinámico [153]. La vista estructural debe encargarse de la definición de los servicios y sus operaciones [157]. Por su parte, la vista dinámica permite el control del flujo de la información por medio del intercambio de mensajes y el uso de los estados de los roles o participantes [157].

Aunque el énfasis es esta tesis doctoral no está en los aspectos de composición de WS, consideramos que el análisis (ver sección 3.3.2) y la formalización (ver sección 3.4.3) que hacemos en el capítulo 3 de los flujos de mensajes de BPMN, son una contribución para la definición formal de WS.

2.6.4 Especificación y modelado de procesos de negocio

El *modelado de negocios* es la técnica por excelencia para alinear los desarrollos con las metas y los objetivos de las empresas e instituciones [67, 120]. Si se realiza de tal forma que el modelo quede consensuado entre los grupos interesados, las posibilidades de éxito del proyecto aumentarán de forma significativa. El modelado de negocios, y más específicamente el BPM, es la forma idónea de comunicación con los usuarios de todos los niveles [90].

Desde que a finales de la década de los 80 Hammer [82, 83, 84] propusiera el concepto de *reingeniería de procesos* para mejorar los BP de las empresas, se han propuesto numerosas técnicas para modelar los procesos como mecanismo necesario para comprender la forma de trabajo en una organización y, en su caso, ayudar a encontrar la problemática que en un momento determinado esté impidiendo alcanzar sus objetivos de manera eficaz y eficiente [83]. También se han desarrollado y comercializado herramientas informáticas de modelado de procesos que permiten al usuario obtener una vista estática del proceso en su conjunto y, en algunos casos, una vista dinámica mediante la simulación de la ejecución del proceso [83, 221].

Tal como indicamos en los trabajos relacionados (ver sección 1.2), la mayoría de los lenguajes de modelado y especificación de BP se centran en el control de flujo. Entre muchas de las técnicas que se han desarrollado para el modelado y la especificación de BP, están: diagramas de actividad de UML, UML Profile for *Enterprise Distributed Object Computing* (EDOC) Business Processes, IDEF, *e–business XML* (ebXML) *Business Process Specification Schema* (BPSS), *Activity–Decision Flow* (ADF) Diagram, RosettaNet, LOVeM y *Event–Process Chains* (EPC) [158]. A continuación se enumeran y explican brevemente algunas de las técnicas y lenguajes de la IS más significativos en el BPM. Posteriormente, en la próxima sección, se explican con mayor detalle los lenguajes y formalismos que han tenido mayor impacto en el BPM.

- Diagrama de Flujo (*Flow Chart*). Datan de los años 60 [191] y se definen como una representación gráfica de una secuencia lógica de procesos de trabajo.
- Diagrama de Flujo de Datos (*Data Flow Diagram*, DFD). Son representaciones de información a través de entidades externas, pasos internos de procesado y elementos de almacenamiento de datos de un proceso de negocio [113]. Estos diagramas permiten ver cómo fluyen los datos a través de la organización y los procesos, así como las transformaciones que sufren dichos datos y los diferentes tipos de salidas. No modela representaciones de flujos de materiales, recursos humanos y otros elementos relacionados con los procesos [225].
- Diagrama Entidad–Relación (*Entity–Relationship* (ER) *Diagram*). Son modelos de red que describe con un alto nivel de abstracción, la distribución de datos almacenados en un sistema [77].
- Diagrama Transición de Estado (*State Transition* (ST) *Diagram*). Se originan para la descripción de la perspectiva dinámica de sistemas dependientes en el tiempo. Son muy útiles ya que proporcionan información explícita acerca de la secuencia de eventos dentro del sistema a lo largo

del tiempo. La limitación está en que no describe la colaboración entre los objetos que causan dichas transiciones.

- Definición Integrada para la Función de Modelado (*Integrated Definition for Function Modelling*, IDEF). Es una familia de técnicas de modelado que ofrecen una perspectiva integrada para representar y modelar procesos y estructuras de datos [131]. La familia IDEF[4], consiste en un gran número de técnicas, entre las cuales se destaca IDEF0 e IDEF3, que son aquellas relacionadas con los BP, aunque existen otras versiones como IDEF1, IDEF1X, IDEF2, IDEF4 e IDEF5.
- Diagrama de Actividad de Rol (*Role Activity Diagram*, RAD). Los RAD son utilizados para esquematizar las actividades bajo la responsabilidad de cada rol, así como la interacción entre ellos y con sucesos externos, entendiendo por rol, el comportamiento deseado de los individuos dentro de la organización [98].
- Diagrama de Interacción de Rol (*Role Interaction Diagram*, RID). Son gráficos que representan los roles de los BP. Su mejor uso se centra en el diseño del flujo de trabajo y suelen ser utilizados para procesos que implican la coordinación de actividades interrelacionadas [8].
- Autómata (*Automaton*). Son un modelo público y la base de las especificaciones formales para los sistemas [94]. Un autómata se compone de un conjunto de estados, acciones, las transiciones entre estados y un estado inicial. Las etiquetas indican la transición de un estado a otro. Muchos de los modelos de especificaciones para expresar el comportamiento de sistemas se derivan de los autómatas [22, 94].
- Redes Petri (*Petri Nets*, PN). Creadas por C.A. Petri en 1962, representan una alternativa para modelar el comportamiento y la estructura de un sistema. Este lenguaje es una generalización de la teoría de autómatas para expresar fácilmente el concepto de eventos concurrentes [155].
- Técnica Orientado a Objeto (*Object–Oriented* (OO) *Technique*): Se utiliza para modelar y programar procesos caracterizados como objetos, que son desarrollados y transformados por actividades y operaciones. Existen diversidad de técnicas basadas en la programación orientada a objetos, pero de todas ellas, la más importante es UML [157].

El concepto más relacionado entre la especificación y el modelado de BP, es el de workflow, que hace referencia a la automatización total o parcial de uno o varios procesos, tal como indicamos en la sección 2.6.1. Los WfMS son aplicaciones informáticas que permiten la definición (modelado) de los procesos que constituyen un flujo de trabajo, y la gestión de su ejecución mediante la interpretación de los modelos y la ejecución de las aplicaciones necesarias para la elaboración de los documentos implicados en el trabajo o

[4] `http://www.idef.com/`

para la intercomunicación de los participantes en el mismo [219]. Se trata, en definitiva, de automatizar el trabajo en una organización de tal forma que un usuario que se conecte al sistema de workflow sepa en todo momento el siguiente trabajo a realizar y disponga, en cada instante, de la información necesaria para ello [219]. La preocupación por definir estándares que permitan optimizar la construcción de este tipo de sistemas dió origen a los patrones de workflow propuestos por van der Aalst [1].

Aunque existen muchas técnicas de representación gráfica de BP, el estándar de modelado gráfico de sistemas, UML, es uno de los más utilizados [38]; en particular, los diagramas de actividad han sido objeto de muchos estudios [68]. Adicionalmente, también se está utilizando el Lenguaje de Marcación Extensible (*Extensible Markup Language*, XML[5]) para representar modelos de procesos para dotar a los BP de una definición más precisa, facilitar su reutilización y garantizar el intercambio de información entre herramientas. El surgimiento de WS–BPEL [153] es muestra de ello. Finalmente, el esfuerzo de la BPMI, con el soporte de la OMG, se ha visto cristalizado con la aparición y consolidación de BPMN [158] como estándar de facto para el BPM.

2.6.5 Lenguajes y formalismos aplicados en el modelado de procesos de negocio

Debido a la naturaleza compleja y dinámica de las organizaciones, los modelos basados en lenguajes formales son necesarios para el análisis del comportamiento y el diseño de BP, y de los sistemas que los implementen, así como para el mejoramiento del funcionamiento de los existentes. A continuación se presentan y explican brevemente los lenguajes y formalismos de la IS más aplicados en el BPM.

2.6.5.1 Redes de Petri

Las PN han sido usadas como herramienta para el modelado y el análisis de BP de las organizaciones [60]. Por una parte, se pueden utilizar como un lenguaje de diseño para la especificación de workflow complejos, y por otra, la teoría de PN proporciona una potente herramienta de análisis para verificar la corrección de los procedimientos workflow [2, 60]. Las PN son útiles por su semántica formal natural de representación y por su expresividad [5]. La PN son utilizadas para modelar BP donde se requiere especificar

[5] `http://www.w3.org/TR/xml/`

qué actividades deben ejecutarse y en qué orden [60]. Permiten definir los procesos de workflow para modelar las actividades como transiciones, las condiciones como estados y las instancias como tokens [2].

Una PN que modela un proceso de workflow se denomina WF–net [205]. Una WF–net tiene un estado inicial de entrada y un estado final de salida. Un token en el estado inicial representa una instancia que debe ser ejecutada. Un token en el estado final representa una instancia que ha sido ejecutada. En una WF–net cada actividad (transición) y condición (estado) debe contribuir al procesamiento de las instancias, cada transición y estado debe estar en el camino desde el estado inicial al estado final. Las redes de workflow se utilizan para diseñar workflow, definir modelos de procesos y analizar su flujo, y permiten representar el *routing* —la dinámica de las transiciones entre los estados— del proceso de workflow. Las WF–net utilizan constructores de routing para recoger la dinámica de los BP, los cuales permiten representar modelos secuenciales, condicionales, paralelos e iterativos [3].

2.6.5.2 Π–calculus

Según la especificación de BPMN [158], la característica más importante de este estándar es que ha sido desarrollado bajo la fundamentación matemática de Π–calculus. Esto permite hacer que los modelos de procesos hechos con BPMN puedan ser ejecutados directamente, y que el Lenguaje de Modelado de Procesos de Negocio (*Business Process Modelling Language*, BPML) y el Lenguaje de Recuperación de Procesos de Negocio (*Business Process Query Language*, BPQL), utilizados para su ejecución inmediata [158]. BPMN considera que un proceso e–business conducido entre dos socios consta de tres partes: una interfaz pública y dos implementaciones de BP locales (una para cada socio) [158]. La interfaz pública es común a los socios y está soportada por protocolos como ebXML[6], RosettaNet[7] y BizTalk[8]. La implementación primitiva es específica de cada socio y es descrita en algún lenguaje de ejecución como BPML. Una vez desarrollada la implementación de los BP locales de un proceso e–business, ésta debe ser implantada en una plataforma que lo ejecutará. Para este propósito se definió BPQL, una interfaz estándar de administración para la implantación y ejecución de procesos e–business. Además, BPQL se basa en *Universal Description, Discovery and Integration* (UDDI)[9] para proveer una forma estándar de registrar, anunciar y descubrir las interfaces públicas de procesos e-business.

[6] `http://www.ebxml.org/`

[7] `http://www.rosettanet.org/`

[8] `http://www.microsoft.com/biztalk/en/us/default.aspx`

[9] `http://uddi.xml.org/`

2.6.5.3 UML

UML [157] preescribe un conjunto de notaciones y diagramas estándar para modelar sistemas orientados a objetos y describe la semántica esencial de lo que estos diagramas y símbolos significan. UML es una consolidación de muchas de las notaciones y conceptos más usadas orientados a objetos [67, 101]. Cuenta con un perfil orientado al BPM, el perfil UML EDOC[10]. Este perfil permite expresar dependencias complejas de datos y temporales de interacciones entre actividades del negocio, iteración de actividades, alternativas de entradas y salidas de las actividades, y funciones relacionadas con los ejecutores, los artefactos y los responsables de las actividades [67, 101].

En particular, para mostrar de forma más detallada el flujo de trabajo que realiza cada BP con UML [157], se utilizan los *diagramas de actividades* con carriles (*swimlanes*), que se llaman *diagramas de proceso* en el ámbito de los BP. Los diagramas de actividad son diagramas de flujo del proceso multi–propósito que se usan para modelar el comportamiento de un sistema o un BP. Son parecidos a los diagramas de flujo; la diferencia clave es que los diagramas de actividad o proceso pueden mostrar procesos en paralelo, lo cual es importante cuando se usan para modelar BP que se desarrollan en paralelo o para modelar la colaboración de participantes dentro de un BP [67]. Se define un carril por cada rol participante en el escenario descrito por el BP, que incluye las actividades que realiza dicho rol. El diagrama también muestra la información que necesita y produce cada actividad, y la sincronización requerida entre las diferentes actividades [67].

Los datos aparecen como objetos que fluyen entre las actividades y pueden tener un estado. Estos objetos son referidos como objetos de información. Así, durante la descripción de un BP mediante un diagrama de proceso, es posible encontrar una actividad cuya complejidad amerite que sea necesario describirla mediante otro diagrama de proceso adicional. Por tanto, este nuevo diagrama de proceso describirá un sub–proceso en relación con el objetivo relacionado con el BP original. De este modo los BP se organizan jerárquicamente. También es posible mostrar en diferentes diagramas de proceso el flujo normal y los flujos alternativos del BP [67, 157].

Para la descripción de las actividades dentro de un BP, se utilizan los *modelos de casos de uso de negocio*, los cuales reflejan gráficamente las metas y funciones que persigue el negocio [101]. Se usan como una entrada esencial para identificar roles y entregables en la organización. Muestra las funciones de negocio para una organización, actores del negocio, trabajadores

[10] `http://www.omg.org/technology/documents/formal/edoc.htm`

del negocio y las interacciones entre ellos. Cada caso de uso describe las actividades (o sub–procesos) que se realizan en la ejecución de un BP [101].

Por su parte, los *diagramas de estado* se usan en el BPM para modelar el comportamiento dinámico de un objeto de negocio en particular, o de una clase de objetos de negocio [67]. Un diagrama de estado se modela para todas los elementos de negocio que se consideran con un comportamiento dinámico; por ejemplo, un trabajador o participante de un BP. En él se modelan la secuencia de estados que el objeto de negocio atraviesa durante una instancia de un BP en respuesta a los estímulos recibidos, junto con sus propias respuestas y acciones. Por ejemplo, el comportamiento de un participante se modela en términos de en qué estado está inicialmente, y a qué estado cambia cuando recibe un evento en particular (por ejemplo, una solicitud de un pedido). También modela qué acciones realiza un partipante en un estado en concreto.

Los *diagramas de secuencia de negocio* se usan para mostrar la secuencia de la lógica de un BP, el orden en que se suceden los mensajes y la interacción de participantes de un BP para realizarlo [101]. Este diagrama es muy importante, especialmente cuando se trabaja en entornos altamente compartidos como lo pueden ser los BP e implementaciones WS. Ayudan a analizar la interacción temporal y sincronización entre participantes y entidades del negocio. Los *diagramas de objetos de negocio* describen los objetos (trabajadores y entidades) y sus relaciones en algún momento. Permiten analizar la criticidad de una entidad de negocio o trabajador dentro del BP; su dependencia con respecto a otros trabajadores [101].

En [38] presentamos la verificación de un BP modelado con el perfil de UML EDOC, donde mostramos el uso de los diagramas descritos anteriormente y describimos cómo se puede analizar el comportamiento de un BP usando el cálculo de procesos CSP+T y la herramientas de MC FDR2 [71].

2.6.5.4 IDEF

Los orígenes de las técnicas IDEF se remontan a la década de los 70' por la necesidad de las Fuerzas Armadas de los Estados Unidos de América para mejorar sus operaciones de producción, iniciándose así el programa *Integrated Computer–Aided Manufacturing* (ICAM). Según [221], la familia IDEF, como herramientas de modelado, en comparación con otras, presenta las siguientes características:

1. *Estandarización*. Define estrictamente el lenguaje gráfico utilizado en el proceso de modelado. Las fases de análisis y los símbolos gráficos hacen que el proceso de modelado sea estándar y conciso. Al mismo tiempo, es-

tablece la plataforma de la cooperación del equipo de desarrollo de software en el análisis de un sistema complicado.
2. *Descripción*. Dado que los gráficos sólo pueden expresar una información limitada, los suplementos del método IDEF describen cualitativamente declaraciones particulares para analizar los detalles. Además, puede mostrar los hechos, las restricciones, y las relaciones de la lógica implícita en los modelos para tener una comprensión sintética de los modelos construidos.
3. *Diálogo*. A diferencia de las técnicas de modelado tradicional, con IDEF los analistas pueden continuamente mantener discusiones con los otros stakeholders del desarrollo desde el inicio de la especificación de requisitos, especialmente en la etapa de evaluación, para evitar inconsistencias en el proceso de modelado. Así que el tiempo de modelado se acorta.

En particular, la técnica IDEF0 está diseñada para modelar las decisiones, acciones y actividades de una organización u otro sistema, y representa la perspectiva funcional de modelado, es decir, el qué [131]. IDEF0 utiliza solo un tipo de anotación en sus representaciones gráficas conocido como *Input–Control–Output–Mechanism* (ICOM). La representación estática de sus diagramas no permite visualizar las perspectivas de modelado de comportamiento o informacional. Para vencer dichas limitaciones, se desarrolló IDEF3 (*Process Description Capture*), que describe a los procesos como secuencias ordenadas de hechos o actividades, representando el cómo, y mostrando la visión dinámica o de comportamiento. IDEF3 es una técnica que hace referencia al modelado de workflow, basado en DFD [225] y que es usado para describir gráficamente las actividades que se siguen dentro de las funciones de un sistema, el camino que sigue la información de una manera ordenada, la relación exacta entre los procesos y, lo más importante, los elementos u objetos que son parte del mismo [131]. Básicamente, IDEF3 sirve para modelar el flujo de los procesos representados en IDEF0, por lo cual son lenguajes que son combinables. El lenguaje es sencillo y muy completo [131, 221].

2.6.5.5 Basados en XML: XPDL, BPML, BPEL4WS

El *XML Process Definition Language* (XPDL)[11] es un formato estandarizado por *Workflow Management Coalition* (WfMC)[12] para almacenar y permitir el intercambio de diagramas de BP. Intenta ofrecer una manera estándar para representar procesos de tal manera que puedan ser importados/exportados

[11] `http://www.xpdl.org/`
[12] `http://www.wfmc.org/`

por cualquier editor que implemente el estándar (actualmente, más de 80 productos lo han implementado). XPDL 2.1 tiene extensiones que permiten representar todos los aspectos de BPMN. XPDL es actualmente el mejor formato del archivo para el intercambio de diagramas BPMN. Se ha diseñado específicamente para almacenar todos los aspectos de un Diagrama de Proceso de Negocio (*Business Process Diagram*, BPD). XPDL contiene elementos para llevar a cabo la información gráfica, tal como la posición X y Y de los nodos, así como los aspectos ejecutables que serían utilizados para el funcionamiento de un BP. Suele ser preferido cuando se trata de implementar procesos o workflows con interacciones humanas, ya que en esto supera a WS–BPEL. Para soportar interacciones humanas, BPEL necesita de estructuras complementarias a la especificación, lo cual dificulta su portabilidad ya que suelen ser implementaciones propietarias de cada proveedor [153].

BPML es un meta–lenguaje para el BPM, así como XML es un meta–lenguaje para el modelado de datos empresariales. BPML proporciona un modelo abstracto de ejecución para la colaboración de BP y procesos transaccionales basados en el concepto de máquina de estados finitos transaccionales [150]. El BPML de más reciente aparición es WS–BPEL, creado en un esfuerzo de colaboración de BEA, IBM, Microsoft y Sun [150].

WS–BPEL es un lenguaje basado en XML que permite especificar el comportamiento de un BP basado en interacciones con WS. La estructura de un proceso WS–BPEL se divide en 4 secciones [153]: (1) definición de relaciones con los socios externos, que son el cliente que utiliza el BP y los WS a los que llama el proceso; (2) definición de las variables que emplea el proceso; (3) definición de los distintos tipos de manejadores que puede utilizar el proceso: manejadores de fallos y de eventos; y (4) descripción del comportamiento del BP; esto se logra a través de las actividades que proporciona el lenguaje. Los principales elementos constructivos de un proceso WS–BPEL son las actividades, que pueden ser de dos tipos: básicas y estructuradas. Las básicas son las que realizan una determinada labor, mientras que las estructuradas pueden contener otras actividades y definen la lógica de negocio. A las actividades pueden asociarse un conjunto de atributos y de contenedores. Estos últimos pueden incluir diferentes elementos, que a su vez pueden tener atributos asociados. Además, WS-BPEL permite realizar acciones en paralelo y de forma sincronizada [153].

2.6.5.6 Otros: Statemate, EPC, AMBER, STRIM

Statemate [85] es un entorno de desarrollo gráfico que permite la especificación, el análisis, el diseño y la documentación de sistemas reactivos grandes y complejos, como son los sistemas empotrados de tiempo real,

sistemas de control y comunicación, y software interactivo. Permite a los usuarios preparar, analizar y depurar de forma gráfica, pero con precisión, las descripciones del sistema bajo desarrollo, a partir de la captura de tres interrelacionados puntos de vista: estructura, funcionalidad y comportamiento. Según [85, 86] Statemate combina dos principios, o tesis, que se consideran son una guía para el diseño de herramientas para el soporte del desarrollo de sistemas. La primera es la necesidad de un amplio soporte a especificaciones ejecutables, y la segunda es la ventaja del uso de formalismos visuales. Sus principales aportaciones son [185]: (a) la integración de lenguajes de modelado tales como diagramas de actividad, diagramas de módulos y diagramas de estados (*statecharts*); (b) la concepción de un sistema descrito a base de formalismos visuales; y (c) la base transformacional en sus producciones. Statemate, representa una excelente base para comprender y desarrollar sistemas reactivos [86] como lo son los BP; específicamente, workflow [212].

Los diagramas EPC son una técnica de BPM, principalmente utilizada para el análisis de procesos con la intención de implementar sistemas de Planificación Empresarial de Recursos (*Enterprise Resource Planning*, ERP) [2, 80]. Los sistemas ERP son usados para gestionar y coordinar todos los recursos, la información y las funciones de un negocio. Por esto se puede decir que los diagramas EPC quedan dentro de la categoría de herramientas para BPM. El *EPC Markup Language* es el formato de XML para el intercambio de modelos EPC, independiente de herramientas y plataformas [2, 80].

El *Architectural Modelling Box for Enterprise Redesign* (AMBER) [63] es un lenguaje de modelado de BP que permite representar gráficamente los procesos, los datos, la organización y las personas involucradas, de una manera integrada y uniforme. En [102] se define una semántica formal en *Process or Protocol Meta Language* (PROMELA) que permite verificar BP con Spin [93].

En [162] se define un lenguaje gráfico, el *Systematic Technique for Role & Interaction Modelling* (STRIM), centrado en el modelado con los RAD de los distintos tipos de actores y en la colaboración entre ellos. Según [154], permite adquirir información a partir de las operaciones con el fin de modelar las decisiones que conducen a cambios de estado de los participantes. El uso de la información se muestra como un conjunto de casos de uso que describe cómo cada participante apoya los diferentes aspectos del proceso [154]. No define formalmente su semántica [69].

2.6.6 Verificación vs. validación de procesos de negocio

En general, la *verificación* se enfoca en el tema de la consistencia interna de un modelo, mientras que la *validación* está relacionada con la correspondencia entre el modelo y la realidad [172, 192, 196]:

Validación. Proceso de comparar la salida del modelo del BP con el comportamiento organizacional. En otras palabras: comparar la ejecución del modelo del BP con la realidad (física u otra cualquiera).

Verificación. Proceso de comparar la implementación de un BP (puede ser un programa o una especificación formal detallada) con el modelo del BP para garantizar que la implementación es correcta con respecto al modelo del BP.

El término validación se aplica a aquellos procesos que buscan *determinar si una implementación es correcta o no respecto al sistema "real"*; por ejemplo, pruebas funcionales o de aceptación. De forma más sencilla, la validación trata sobre la cuestión *¿se está construyendo/modelando el BP correcto?*, mientras que la verificación responde a *¿se está construyendo correctamente el BP?*. La verificación *comprueba que la implementación del modelo (programa o especificación formal detallada) corresponde al modelo del BP*, mientras que la validación *comprueba que el modelo del BP corresponde con la realidad* [172, 192, 196].

En este trabajo nos centramos en la verificación formal, ya que por las características de los BP con criticidad y de los sistemas que los soportan, *es sumamente costoso y riesgoso esperar que el sistema que soporta al BP esté implementado para validar que la implementación cumple con lo que se quiere en la realidad*. Por otro lado, también es parte de nuestros objetivos *soportar al ingeniero de BPM con métodos, modelos y herramientas que le permitan verificar los modelos de los BP que construye, desde el mismo momento que los define y los diseña*.

2.6.7 Soporte de la verificación para el análisis de procesos de negocio

Se han desarrollado varios trabajos que permiten la verificación y el análisis de BP utilizando los lenguajes y formalismos mencionados a lo largo de este capítulo. Tal como indicamos en los trabajos relacionados (ver sección 1.2), consideramos como los más importantes y con estrecha relación con lo que proponemos en esta tesis doctoral, los trabajos referidos a: (1) Woflan [206, 207] y WF–nets [205], (2) Semántica y verificación de diagramas de

actividad UML [68], (3) Modelo para la verificación formal de BP [69] y VERBUS [70] y (4) Enfoque algebraico de procesos y formalizaciones de BPMN [214].

Adicionalmente, en [151] se presenta un estudio que examina los esfuerzos de la aplicación de la verificación formal en BPM. El autor presenta las técnicas de verificación formal (autómatas, PN y álgebras de proceso) que permiten simular y verificar modelos gráficos de BP en la fase de diseño de los sistemas de información empresariales; y reitera que estas técnicas pueden detectar y corregir errores de los modelos en etapas tempranas del desarrollo; antes de la implementación. Por otra parte, indica que hay estudios, por ejemplo, [115, 223], para verificar diagramas de máquinas de estado de UML con MC, que deben ser tomados en cuenta.

Por nuestra parte, hemos realizado algunos aportes orientados al diseño y verificación de sistemas con criticidad, reflejados en [21, 36, 39, 139, 143]. Como productos parciales de esta tesis doctoral, en cuanto a la verificación composicional de BP, están los trabajos [37, 38, 140]. Todos estos aportes nos han permitido conformar un *Enfoque Formal de Verificación Composicional*, el EFVC (descrito en el capítulo 4). Este enfoque integra la verificación composicional y la técnica de MC, aplicables a la verificación de BP.

Hecha la revisión de los aspectos conceptuales y teóricos que conforman el Estado del Arte de este trabajo, en el próximo capítulo describiremos la base formal que soporta la propuesta que hacemos a través de la presente tesis doctoral.

Capítulo 3
Base Formal

Resumen El objeto de este capítulo es presentar los formalismos que soportan las aportaciones que se desarrollan a lo largo de la tesis doctoral. En la sección 3.1 describimos el modelo de tiempo en el cual se basa nuestra propuesta. Luego, en la sección 3.2, damos una definición de autómatas, así como de autómatas temporizados y de autómatas de Büchi, los cuales son los formalismos que soportan la validación de nuestro esquema de verificación. Seguidamente, en la sección 3.3, presentamos la semántica formal temporal que proponemos para los elementos notacionales de tiempo de BPMN A continuación, en la sección 3.4, damos la definición de *regla de transformación* como preámbulo a la descripción de las ideas que sustentan los sistemas de reglas de transformación utilizados en nuestra propuesta, donde se definen las reglas de transformación que nos permiten expresar tanto especificaciones en CCTL como modelos de BP en BPMN y de sistemas en UML–RT, en el lenguaje de especificación formal CSP+T. Finalmente, en la sección 3.5, se expone la tecnología que sirve de base para implementar el esquema de verificación composicional que proponemos.

3.1 Modelos de tiempo

Al especificar y modelar el comportamiento temporal de los sistemas, en general, y de los BP con restricciones temporales, en particular, hay que decidir la manera de modelar el tiempo. Para ello, se pueden utilizar alguno de los siguientes modelos de tiempo [129]:

- *Continuo.* Este modelo permite capturar el tiempo de respuesta a estímulos con una buena precisión, ya que supone que entre dos instantes de tiempo t_1 y t_2, siempre exite otro instante de tiempo t.
- *Discreto.* En los sistemas de computación el tiempo se mide por relojes, los cuales dividen un intervalo continuo de tiempo en una secuencia finita de instantes, con lo cual cada instante corresponde a un *tick* del reloj. Entre dos *ticks* seguidos no se puede capturar ningún instante de tiempo. De esta manera, el intervalo entre ellos es ingnorado por el sistema. Esto representa que entre dos instantes seguidos medidos por un reloj no existe otro instante de tiempo medido por el mismo reloj [103].

Con la Definición 3.1 podemos transformar un intervalo de tiempo continuo a otro intervalo de tiempo discreto, de tal manera que un intervalo de tiempo continuo se convierta en una secuencia finita de instantes de tiempo. De esta manera, la precisión temporal de un sistema depende de las unidades de medida que se utilizan en él. Denotamos con $\mathbb{R}^+$ al conjunto de los números reales positivos y con $\mathbb{N}_0$ al conjunto de los números naturales incluyendo al cero.

Definición 3.1. Tiempo discreto. Sean $t \in \mathbb{R}^+$, $t_n \in \mathbb{N}_0$, $\varepsilon \in [0,1)$, $t_n \leq t \leq t_n + \varepsilon$ y la constante $k \in [0,1)$. El valor de t cambia al valor t_n, precedente o siguiente, según la función:

$$\Phi(t) = \begin{cases} t_n & \text{si } \varepsilon < k \\ t_n + 1 & \text{si } \varepsilon \geq k \end{cases} \tag{3.1}$$

Para nuestro enfoque, tomamos como modelo de tiempo al discreto y como dominio de tiempo al conjunto $\mathbb{N}_0$. La secuencia de tiempo $\tau = \tau_1, \tau_2, \ldots$, es una secuencia infinita de valores de tiempo, $\tau_i \in \mathbb{N}_0$, que satisface las siguientes propiedades:

1. *Monotonía.* τ crece de manera estrictamente monótona: $\forall\, i \geq 1 \Rightarrow \tau_i < \tau_{i+1}$.
2. *Progreso.* $\forall\, \tau \in \mathbb{N}_0, \exists\, i \geq 1 / \tau_i > \tau$.
3. *Tiempo de Newton.* El tiempo progresa en todos los procesos con la misma velocidad y con el mismo y único marco de tiempo global.

3.2 Representación de estructuras de Kripke

El recurso más utilizado actualmente en la verificación del modelo de un sistema es el uso de un autómata que implementa directamente una EK [16, 22]. En las próximas secciones trataremos los aspectos más importantes de los autómatas y los autómatas temporizados, hasta presentar el tipo de autómata temporizado que utilizamos en este trabajo.

3.2.1 Autómatas

Los autómatas son modelos matemáticos de un *sistema* con *entradas* y *salidas*, pudiendo estar en cualquiera de un número finito de *estados*. Cada estado establece la relación entre entradas anteriores para alcanzar ese estado y las entradas posteriores (salidas) para alcanzar un estado siguiente [94].

Formalmente, se define un autómata $\mathbf{A}_M$ (ver Definición 3.2) que acepta exactamente las secuencias (finitas) de las valoraciones en un camino a través de la EK, como sigue:

Definición 3.2. Autómata. Sea $M = \langle S, s_0, R, L\rangle$ una EK sobre un conjunto de proposiciones atómicas, AP. Un autómata $\mathbf{A}_M$ es una tupla $\langle \Sigma, S_M, S_M^0, \Delta_M, F_M\rangle$, donde

- $\Sigma = 2^{AP}$,
- $S_M = S \cup \{s^i\}$,
- $S_M^0 = \{s^i\}$,
- $\forall\, s, s' \in S_M, a \in \Sigma : \langle s, a, s'\rangle \in \Delta_M$ si y sólo si $L(s') = a$ y $((\langle s, s'\rangle \in R)$ o $(s = s^i$ y $s' = s^0))$; y
- $F_M = S_M$.

Comparando la Definición 3.2 anterior con la definición de las EK (ver Definición 2.10), existe una relación muy cercana entre éstos. Según [11, 16] los cambios son los siguientes:

- el etiquetado en las EK es sobre los estados en lugar de tener las etiquetas en los arcos,
- el etiquetado en los autómatas consiste en un elemento de Σ en lugar de un subconjunto de AP, y
- en las EK no existe una definición de estados finales (se puede pensar que todos los estados podrían ser finales).

3.2.2 Autómata temporizado

El concepto de *Autómata Temporizado* (AT) fue propuesto por primera vez en [11] como una extensión de la teoría de autómatas enfocada al modelado de sistemas en tiempo real [94]. Un AT (ver Definición 3.3) acepta *términos temporizados*; es decir, la ocurrencia de un valor en tiempo real está asociada con cada símbolo. En concreto, un AT es una máquina en estado finito equipada con un conjunto de *relojes* [11, 94]. Los relojes son funciones continuas a trozos de valores reales (un modelo de tiempo denso) que registran con precisión el tiempo transcurrido entre eventos. Los relojes están habilitados para reiniciarse con un nuevo valor, el cual se convierte en el valor inicial de la siguiente transición. Las transiciones están asociadas con una *guarda* (restricción temporal) la cual es un predicado sobre los relojes. La guarda determina cuando ocurre una transición.

Definición 3.3. Autómata temporizado. Un *Autómata Temporizado* (AT) **A**, es una tupla $\langle \Sigma, S, I, C, R \rangle$, donde Σ es un alfabeto de entrada, S es un conjunto finito de estados, $I \subseteq S$ es el conjunto de estados iniciales, C es un conjunto finito de relojes y $R \subseteq S \times S \times \Sigma \times 2^C \times \Phi(C)$ provee el conjunto de transiciones. Un arco $\langle s, s', a, \lambda, \delta \rangle \in R$ representa una transición desde el estado s al estado s' activada por el símbolo de entrada a. El conjunto $\lambda \subseteq C$ provee los relojes que se reinician con la transición, y δ es una restricción temporal sobre C.

Un término $(string, secuencia)$ temporizado sobre un alfabeto Σ es un par $(\bar{\sigma}, \bar{\tau})$ donde $\bar{\sigma}$ es un término infinito y $\bar{\tau}$ es una secuencia de tiempo. Para un término temporizado $(\bar{\rho}, \bar{\tau})$, una ejecución $r = (\bar{s}, \bar{v})$ de un autómata temporizado es una secuencia infinita $r : \langle s_0, v_0 \rangle \xrightarrow{\sigma_1, \tau_1} \langle s_1, v_1 \rangle \xrightarrow{\sigma_2, \tau_2} \langle s_2, v_2 \rangle \xrightarrow{\sigma_3, \tau_3} \ldots$, donde $s_0 \in S_0$, para todo $t \in C$, $v_0(t) = 0$, y existe un arco $\langle s_{i-1}, s_i, \sigma_i, \lambda_i, \delta_i \rangle \in R$ para todo $i \geq 1$ tal que $(v_{i-1} + \tau_i - \tau_{i-1})$ satisface δ_i y $v_i = [\lambda_i \rightarrow 0](v_{i-1} + \tau_i - \tau_{i-1})$. Para tal ejecución, $inf(r) = \{s \in S \mid s = s_i\}$ para un número infinito $i \geq 0$ de ejecuciones$\}$.

3.2.3 Autómata de Büchi

Diferentes clases de ω–autómatas están definidos por añadir una condición de aceptación (es decir, *estados de aceptación*[1] en lugar de estados finales) a la definición de las relaciones de transición (p.e., Büchi, Muller, Rabin). Nosotros usamos el Autómata de Büchi Temporizado (*Timed Büchi Automaton*, TBA) (ver definición 3.4) ya que éste es el autómata más simple sobre términos infinitos [49] y nos permite representar procesos regulares de tiempo [41].

Definición 3.4. Autómata de Büchi temporizado. Un *Autómata de Büchi Temporizado* (Timed Büchi Automaton, TBA) **A**, es una tupla $\langle \Sigma, S, I, C, R, F \rangle$, donde $\langle \Sigma, S, I, C, R \rangle$ es un autómata temporizado, y $F \subseteq S$ es el conjunto de estados de *aceptación*.

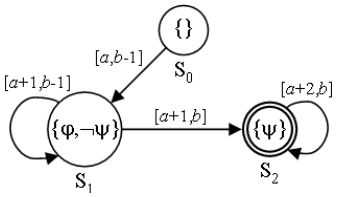
Una ejecución $r = (\bar{s}, \bar{v})$ de un TBA sobre un término temporizado (σ, τ) es *aceptado* si contiene algún estado de aceptación un número infinito de veces; o formalmente, si y sólo si $inf(r) \cap F \neq \varnothing$. F puede estar vacío. En ese caso, todos los términos infinitos que pueden derivarse de una ejecución del autómata son aceptados. Como se puede ver, una ejecución de un TBA **A** sobre un término infinito $(\bar{\sigma}, \bar{\tau}) \in \Sigma^\omega$ es definida casi igual a un recorrido

[1] Los estados de aceptación son aquellos estados de un autómata que no contienen una eventualidad en su siguiente estado; es decir, el estado alcanzado corresponde al final de una secuencia válida para el sistema representado por dicho autómata.

de un autómata finito sobre un término finito, excepto que ahora $\mid v \mid = \omega$ [49].

3.2.3.1 Representación de fórmulas de la Lógica de Árbol Computacional Temporizada como Autómatas de Büchi

Para mostrar el uso de los operadores de CCTL previamente presentados en la sección 2.1.4, describiremos una fórmula CCTL ϕ, la cual establece que la fórmula ψ se hará verdad dentro del intervalo $[a, b]$ y para todos los pasos de tiempo anteriores, la fórmula φ tiene que ser válida. La especificación de esta fórmula corresponde a lo que se muestra en la Fig. 3.1, tanto a través de un TBA como textualmente. Los estados con doble línea representan a los estados de aceptación del TBA, que corresponden a los estados de satisfacción con éxito de la fórmula CCTL (Fig. 3.1) y del término de proceso CSP+T (Fig. 3.2), respectivamente.

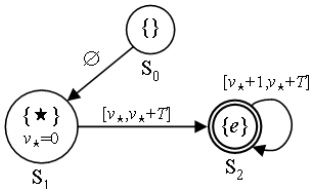


$$\phi = \varphi \mathsf{U}_{[a,b]} \psi$$

Fig. 3.1 Ejemplo de una fórmula CCTL y su representación a través de un TBA.

3.2.3.2 Representación de términos de Procesos Secuenciales de Comunicación + Tiempo como Autómatas de Büchi

Para mostrar el uso de los operadores de CSP+T previamente presentados en la sección 2.2.3.2, describiremos un término de procesos CSP+T P, el cual sólo puede implicarse en el evento e, el cual puede ocurrir sólo un máximo de T unidades de tiempo desde el instante de tiempo de instanciación del proceso $\star$ (evento precedente), registrado en la *variable marcadora* $v_\star$. La especificación de este proceso corresponde a lo que se muestra en la Fig. 3.2, tanto a través de un TBA como textualmente.



$P = 0.\star \bowtie v_\star \rightarrow \mathrm{I}(T, v_\star).e \rightarrow P'$

Fig. 3.2 Ejemplo de un término de proceso CSP+T y su representación a través de un TBA.

3.3 Semántica temporal formal para BPMN

En esta sección presentamos la semántica formal temporal que proponemos para los elementos notacionales de tiempo de BPMN. En el Apéndice A se dan algunos aspectos generales de BPMN en el contexto de la verificación y se hace una presentación general de los elementos notacionales de BPMN.

3.3.1 Elementos notacionales de tiempo en BPMN

Del conjunto extendido de elementos notacionales con los cuales cuenta BPMN para modelar BP (descritos en el Apéndice A), existen algunos que están referidos a aspectos temporales de los BP; a saber: el inicio programado de procesos, los retrasos de procesos, la duración mínima y máxima de actividades, así como la interrupción de una actividad por un flujo de excepción temporizado que se origina por el vencimiento de un plazo de tiempo determinado. Adicionalmente, y de sumo interés para este trabajo, son los aspectos de sincronización de mensajes para lograr una correcta colaboración. Cabe indicar que BPMN no hace ningún tipo de referencia ni especifica nada respecto de las restricciones temporales que suelen existir en la sincronización de procesos concurrentes.

En la Tabla 3.1 se describen brevemente los elementos notacionales temporizados de BPMN y en la próxima sección exponemos un análisis temporal de los mismos y que permite su definición bajo nuestro enfoque.

3.3.2 Análisis temporal de los elementos notacionales de tiempo de BPMN

Con la semántica dada en [218] se logra dar una noción de tiempo que permite coordinar la ejecución de los estados BPMN que se ejecutan concurrentemente en términos del lenguaje de especificación no temporizado CSP.

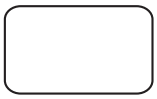
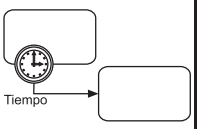
Elemento	Descripción	Notación
Actividad	La realización de actividades conlleva el consumo de una cantidad de tiempo para culminar el trabajo que éstas establecen. Sin embargo, BPMN no contempla atributos relacionados con su duración mínima y máxima. Ya en [214] se incluyen estos atributos como parte de la semántica propuesta, pero no son utilizados para controlar su ejecución.	
Evento de inicio temporizado	Especifica la programación o la fijación de un ciclo (p.e., cada domingo a las 13 hs.) para iniciar el proceso. BPMN no especifica atributos relacionados con el tiempo de la programación o del ciclo; queda como parte de la definición del disparador del evento (*trigger*).	
Evento intermedio temporizado	Especifica la programación o la fijación de un ciclo (p.e., hasta el domingo a las 13 hs.) para disparar el evento como parte del proceso. Si es colocado como parte de un flujo de secuencia, actúa como un mecanismo de retraso. Ver la descripción del *Flujo de excepción temporizado* para cuando es usado para el manejo de excepciones por tiempo. BPMN no especifica atributos relacionados con el tiempo de la programación o del ciclo; queda como parte de la definición del trigger del evento.	
Flujo de excepción temporizado	Cuando un Evento intermedio temporizado es colocado en el borde de una actividad (bien sea una tarea o un sub–proceso) se crea un flujo de excepción por tiempo. La excepción interrumpe la actividad y se manifiesta cuando se cumple el tiempo o el ciclo programado. Si la actividad es completada antes de que se consuma el tiempo especificado por el Evento intermedio temporizado, éste queda sin efecto para el proceso.	
Flujo de mensaje	Modela el intercambio de mensajes entre dos entidades (representadas por dos contenedores separadas en un BPD) que están *preparadas para enviar y recibir mensajes*. Dado que la especificación de BPMN no define claramente qué significa *preparadas*, en este trabajo precisamos el término desde el punto de vista temporal. Establecemos que las entidades deben estar sincronizadas para que el intercambio de mensajes pueda efectuarse. En este sentido, la sincronización requiere definir restricciones temporales que garanticen la comunicación.	

Tabla 3.1 Elementos notacionales de tiempo de BPMN.

Sin embargo, en el ámbito de los BPs, no sólo se requiere que las actividades sean realizadas en el orden correcto y la coordinación de actividades concurrentes no lleve a situaciones de bloqueo o fallo, si no que, además, sean completadas en los tiempos pre–establecidos por los SLA del negocio. Los procesos relacionados con la prestación de servicios típicamente se diseñan tomando en cuenta tiempos de respuesta iniciales y cumplimiento de tiempos máximos. Muchas veces la QoS de los BP es medida en función del tiempo máximo en el cual deben ejecutarse las actividades y del tiempo de duración esperado para el proceso como un todo.

A continuación explicaremos la importancia de dar una semántica temporal precisa a los estados de tiempo BPMN y su relación con las especificaciones temporales que pueden establecerse en documentos del negocio como los SLA o los niveles de QoS. Para efectos de ilustrar de una forma gráfica nuestra semántica, en las Fig. 3.3 y 3.4 hacemos una correspondencia directa entre el tamaño de las figuras que denotan actividades (rectángulos redondeados) y la duración máxima (*ran.max*) y la duración mínima (*ran.min*) que se establece como parte de los atributos de las actividades. Además, denotamos con t_x los tiempos de ocurrencia de los eventos de invocación ε_x de los estados BPMN, y con $S\text{x}.ran.min$ y $S\text{x}.ran.max$ los rangos mínimo y máximo de duración de las actividades Sx, respectivamente, según lo establece BPMN. Adicionalmente, en las Fig. 3.3 (b) a (e) la secuencia de actividades está ordenada sobre líneas de tiempo cuyo origen corresponde con el instante de tiempo de invocación de la actividad $S1$, tal como lo establece BPMN para un *Evento de inicio simple*. Eliminamos de la ilustración las flechas que denotan los flujos de secuencia, ya que por definición su tiempo de duración es cero, según la especificación de BPMN [158]. Para los casos mostrados en la Fig. 3.3, sólo las actividades requieren de una cantidad de tiempo para ejecutarse. Para simplificar, hacemos abstracción acerca de si $S1$, $S2$ y $S3$ son compuestas o atómicas.

3.3.2.1 Actividad

Sea el proceso simple conformado por 3 actividades ($S1$, $S2$ y $S3$) que se ejecutan según el flujo mostrado en la Fig. 3.3. De acuerdo al esquema dibujado en la Fig. 3.3 (a), y a la semántica de BPMN dada en [218], el inicio de la ejecución de la actividad $S2$ (es decir, la ocurrencia del evento ε_{S2}) depende del instante de finalización de la actividad $S1$, el cual debe ocurrir dentro del rango mínimo ($S1.ran.min$) y máximo ($S1.ran.max$) de duración de la actividad $S1$. El inicio de los rangos $S1.ran.min$ y $S1.ran.max$ depende de la ocurencia del evento ε_{S1}. Entonces, el retraso que puede sufrir el inicio de la ejecución de la actividad $S2$ (i.e, la ocurrencia del evento ε_{S2}), varía entre $S1.ran.min$ y $S1.ran.max$. De la misma forma se puede razonar acerca del inicio de la ejecución de la actividad $S3$ (ocurrencia del evento ε_{S3}), el cual varía entre $S2.ran.min$ y $S2.ran.max$, y su dependencia con el inicio de la actividad $S2$ (ocurrencia del evento ε_{S2}).

La duración máxima de la ejecución de un BP (o su peor tiempo de ejecución) corresponde a la suma de las duraciones máximas de todos los estados de tiempo de BPMN presentes en el BP. Como en el BPD de las Fig. 3.3 (a) y (b) sólo tenemos las actividades $S1$, $S2$ y $S3$, podemos afirmar que la especificación inicial $\mathbf{S}_{max}$ del BP es:

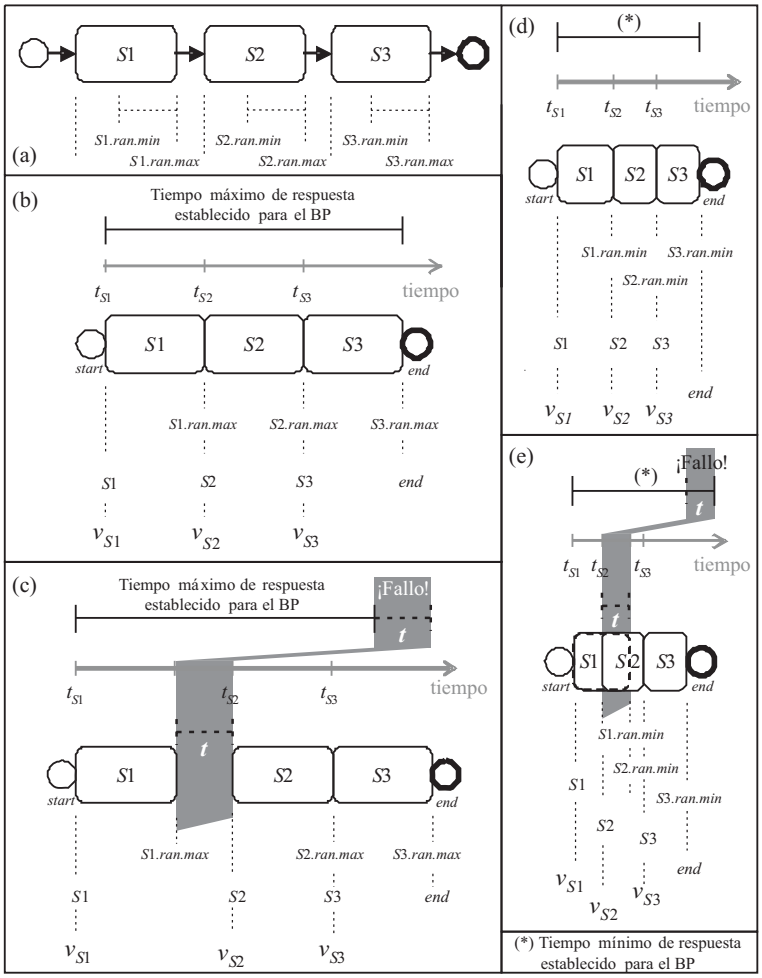


Fig. 3.3 Análisis gráfico del comportamiento temporal de actividades BPMN.

$$\mathbf{S}_{max} = S1.ran.max + S2.ran.max + S3.ran.max = \sum_{i:1..n}^{3} Si.ran.max \tag{3.2}$$

En la Fig. 3.3 (c) mostramos el caso cuando el inicio de la ejecución de la actividad $S2$ (es decir, la ocurrencia del evento ε_{S2}) sucede con un retraso de tiempo t con respecto al tiempo máximo de finalización de la actividad $S1$ ilustrado en la Fig. 3.3 (b). Como se muestra en la Fig. 3.3 (c), ahora el retraso máximo que sufre la ejecución de la actividad $S2$ varía entre $S1.ran.max$ y $S1.ran.max + t$. Si los tiempos relacionados con la

ejecución de las actividades $S2$ y $S3$, no sufren ningún retraso adicional, tenemos que la nueva especificación $\mathbf{S}'_{max}$ del BP es:

$$\mathbf{S}'_{max} = S1.ran.max + t + S2.ran.max + S3.ran.max = \left(\sum_{i:1..n}^{3} Si.ran.max \right) + t \quad (3.3)$$

Análogo al razonamiento anterior, teniendo en cuenta que la duración mínima de la ejecución de un BP (o su mejor tiempo de ejecución) corresponde a la suma de las duraciones mínimas de todos los estados de tiempo de BPMN presentes en el BP, según las Fig. 3.3 (a) y (d) podemos afirmar que la especificación inicial $\mathbf{S}_{min}$ del BP es:

$$\mathbf{S}_{min} = S1.ran.min + S2.ran.min + S3.ran.min = \sum_{i:1..n}^{3} Si.ran.min \quad (3.4)$$

En la Fig. 3.3 (e) mostramos el caso cuando el inicio de la ejecución de la actividad $S2$ (es decir, la ocurrencia del evento ε_{S2}) sucede con un adelanto de tiempo t con respecto al tiempo mínimo de finalización de la actividad $S1$ ilustrado en la Fig. 3.3 (d). Como se muestra en la Fig. 3.3 (e), ahora el retraso mínimo que sufre la ejecución de la actividad $S2$ varía entre $S1.ran.min - t$ y $S1.ran.min$. Si los tiempos relacionados con la ejecución de las actividades $S2$ y $S3$, no sufren ningún adelanto adicional, tenemos que la nueva especificación $\mathbf{S}'_{min}$ del BP es:

$$\mathbf{S}'_{min} = S1.ran.min - t + S2.ran.min + S3.ran.min = \left(\sum_{i:1..n}^{3} Si.ran.min \right) - t \quad (3.5)$$

Podemos ver en las relaciones (3.3) y (3.5), cómo el retraso o el adelanto de tiempo t del inicio de la actividad $S2$, respectivamente, influye en el tiempo máximo o el tiempo mínimo, respectivamente, de duración del BP analizado; es decir, el tiempo de duración del BP se ve incrementado o disminuido, respectivamente, t unidades de tiempo, lo cual no satisface la especificación inicial $\mathbf{S}$ del BP. Para que las relaciones (3.3) y (3.5) cumplan con la especificación original del BP ($\mathbf{S}_{max}$ y $\mathbf{S}_{min}$), se tiene:

$$\mathbf{S}'_{max} \ \textbf{sat} \ \mathbf{S}_{max} \Leftrightarrow t = 0 \quad (3.6)$$

$$\mathbf{S}'_{min} \ \textbf{sat} \ \mathbf{S}_{min} \Leftrightarrow t = 0 \quad (3.7)$$

Lo anterior nos permite afirmar que si no se restringen los intervalos de tiempo en los cuales pueden iniciarse las actividades y no se controla el tiempo mínimo y máximo de duración de las actividades, con seguridad se incurrirá en un fallo al incumplir los tiempos estimados de duración de un

BP (es decir, su especificación original), independientemente de que las actividades se ejecuten correctamente.

Controlar estos tiempos nos permite asegurar que la invocación de las actividades del proceso se haga en el momento oportuno. De esta manera se garantiza que el tiempo de ejecución de las actividades, (1) no sobrepase su rango máximo de duración, y (2) no adelante su rango mínimo de duración. Si cualquiera de estos casos llegase a pasar el proceso debería detenerse y notificar dónde ocurrió el fallo.

Más adelante, en la sección 3.3.2.4, mostraremos que si los tiempos de duración de los elementos de tiempo de BPMN no son controlados, la invocación temprana o tardía de actividades conlleva a que no se logre la sincronización de actividades entre participantes que colaboran de forma concurrente en la ejecución un BP.

3.3.2.2 Evento de inicio temporizado y evento intermedio temporizado

Los estados BPMN *Evento de inicio temporizado* y *Evento intermedio temporizado* [158], son usados normalmente para retrasar la ocurrencia de actividades. De acuerdo a los esquemas de las Fig. 3.4 (a) y 3.4 (b), después de invocar el *Evento de inicio temporizado* y el *Evento intermedio temporizado* a través de los eventos ε_{stime} y ε_{itime}, respectivamente, la invocación de los eventos ε_{S1} y ε_{S2}, respectivamente, ocurre después de finalizado los tiempos de retraso $stime.ran$ e $itime.ran$, respectivamente.

Para el *Evento de inicio temporizado* se cumple:

$$(t_{stime} < t_{S1}) \wedge (t_{S1} \geq t_{stime} + stime.ran)\,, \tag{3.8}$$

donde t_{stime} denota el instante en el cual ocurre el evento ε_{stime} que invoca el *Evento de inicio temporizado* y el inicio del proceso, $stime.ran$ denota el retraso especificado por el *Evento de inicio temporizado*, y t_{S1} denota el instante de tiempo en el cual ocurre el evento ε_{S1} que corresponde a la invocación de la actividad $S1$.

Análogamente, para el *Evento intermedio temporizado* se cumple:

$$(t_{S1} < t_{itime} < t_{S2}) \wedge (t_{S2} \geq t_{itime} + itime.ran)\,, \tag{3.9}$$

donde t_{itime} denota el instante en el cual ocurre el evento ε_{itime} que invoca el *Evento intermedio temporizado*, $itime.ran$ denota el retraso especificado por el *Evento intermedio temporizado*, y t_{S2} denota el instante de tiempo en el cual ocurre el evento ε_{S2} que representa la invocación de la actividad $S2$.

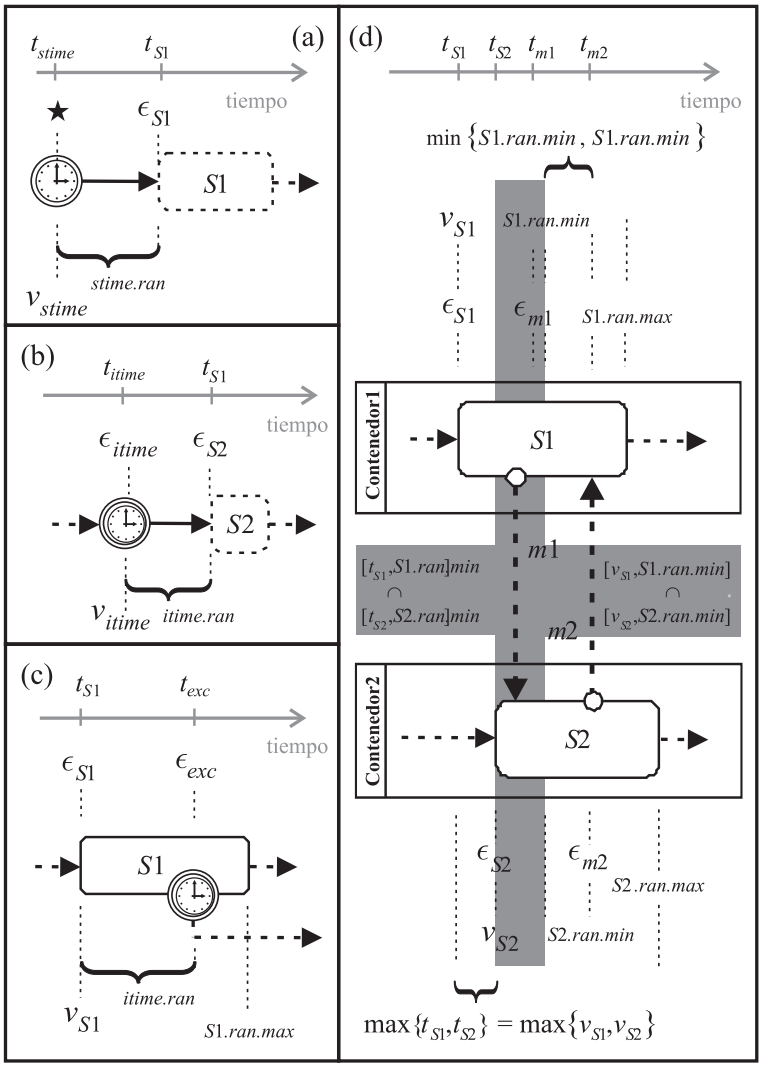


Fig. 3.4 Análisis gráfico del comportamiento temporal de eventos de tiempo y flujos de mensaje de BPMN.

3.3.2.3 Flujo de excepción temporizado

BPMN modela la ocurrencia de cambios en el flujo normal de ejecución de las actividades a través de los *Flujos de excepción* [158]. Estos elementos notacionales permiten modelar la interrupción de la ejecución de una actividad y el cambio de la secuencia del workflow cuando ocurre algo inesperado (un evento —llamado *Evento intermedio*— según BPMN [158]). En la

Fig. 3.4 (c) se ilustra el ejemplo de un *Flujo de excepción temporizado*. En este caso, la ejecución de la actividad $S1$ es interrumpida al pasar el tiempo especificado en $itime.ran$ del *Evento intermedio temporizado* que conforma el *Flujo de excepción temporizado*. La ocurrencia del evento ε_{exc} sucede al transcurrir el lapso de tiempo $itime.ran$ a partir de la ocurrencia del evento ε_{S1} y antes de la finalizar la actividad $S1$; es decir, antes de cumplirse el tiempo $S1.ran.max$.

Del esquema de la Fig. 3.4 (c) se desprende:

$$(itime.ran < S1.ran.max) \wedge (t_{exc} = t_{S1} + itime.ran) \wedge (t_{exc} \in [t_{S1}, S1.ran.max)), \quad (3.10)$$

donde t_{exc} denota el instante en el cual ocurre el evento ε_{exc} que representa la invocación del *Flujo de excepción*, y que interrumpe la ejecución de $S1$. De acuerdo con la relación anterior, una especificación correcta de un *Flujo de excepción temporizado* debe asegurar que la invocación del evento ε_{exc} ocurra dentro del intervalo máximo de duración de la actividad al cual está asociado.

3.3.2.4 Flujo de mensajes

Mientras los *Flujos de secuencia* están restringidos a un *contenedor* individual, los *Flujos de mensaje* representan comunicación entre *contenedores*. Estas comunicaciones deben realizarse dentro de límites de tiempo que permitan asegurar que el BP va a ejecutarse respetando los tiempos establecidos en su especificación original.

Para ilustrar nuestra propuesta en el ámbito de los *Flujos de mensaje*, en la Fig. 3.4 (d) se muestra la colaboración entre dos participantes $Contenedor1$ y $Contenedor2$. La actividad $S1$ envía el mensaje $m1$, el cual es recibido por la actividad $S2$, mientras que la actividad $S2$ le envía el mensaje $m2$ a la actividad $S1$.

Desde el punto de vista temporal, para que la comunicación pueda establecerse tanto la actividad que envía el mensaje como la que recibe el mensaje deben estar preparadas para comunicarse [158]. En concreto, (1) el envío de los mensajes debe hacerse dentro de los límites de tiempo de duración de la actividad que envía, y (2) la recepción de los mensajes debe hacerse dentro de los límites de tiempo de duración de la actividad que recibe. Esto significa que una actividad puede enviar y/o recibir mensajes mientras no haya ocurrido el evento de invocación de la actividad que le precede. Según la Fig. 3.4 (d), estas condiciones se traducen en:

$$\begin{array}{l} 1)\ (t_{S1} < t_{m1} \leq (t_{S1} + S1.ran.max)) \wedge (t_{S2} < t_{m1} < (t_{S2} + S2.ran.min)) \\ \wedge \\ 2)\ (t_{S2} < t_{m2} \leq (t_{S2} + S2.ran.max)) \wedge (t_{S1} < t_{m2} < (t_{S1} + S1.ran.min))\,; \end{array} \tag{3.11}$$

es decir, 1) el mensaje ha de ser enviado por la actividad $S1$ (o $S2$) antes del inicio de la actividad $S2$ (o $S1$, repectivamente) y 2) ha de ser recibido antes de que la actividad $S2$ (o $S1$, respectivamente) pueda terminar. Lo anterior equivale a

$$[t_{S1}, t_{S1} + S1.ran.min] \cap [t_{S2}, t_{S2} + S2.ran.min] \neq \varnothing\,. \tag{3.12}$$

Entonces, de acuerdo al esquema de la Fig. 3.4 (d), para que la comunicación pueda establecerse:

$$t_{m1}, t_{m2} \in [\max\{t_{S1}, t_{S2}\}, \min\{t_{S1} + S1.ran.min, t_{S2} + S2.ran.min\}]\,. \tag{3.13}$$

Este intervalo está representado en la Fig. 3.4 (d) por la zona sombreada que equivale a la intersección de los intervalos $[t_{S1}, S1.ran.min]$ y $[t_{S2}, S2.ran.min]$. Fíjese que la especificación abarca al mismo tiempo la restricción temporal para el envío y la recepción de los mensajes. Esto garantiza que la actividad receptora del mensaje no entre en un estado de bloqueo (en inglés, *deadlock*); es decir, de espera indefinida del mensaje.

3.4 Sistemas de reglas de transformación

En esta sección indicaremos brevemente los aspectos teórico–prácticos que sustentan los conjuntos de reglas de transformación que utilizamos en nuestro enfoque con la finalidad de efectuar la verificación bajo un mismo dominio semántico. Para este trabajo, sobre la base de la definición dada en [160], consideraremos que una *regla de transformación* es *la definición de forma clara y evitando cualquier ambigüedad por parte de quien la usa, de la relación implícita que existe entre los contextos o dominios involucrados en la transformación, y que permite ir de un dominio o contexto a otro*. Brevemente, podemos decir que un conjunto de reglas de transformación establecen la forma para pasar de unas fórmulas "bien formadas' a otras fórmulas 'bien formadas' de un cálculo a otro.

Típicamente, para lograr la automatización de las reglas de transformación se recurre a los algoritmos, ya que son la base fundamental del procesamiento automatizado. En este sentido, dado que nuestra meta final, ara cumplir los objetivos de esta tesis doctoral, es la automatización del pro-

ceso de verificación de sistemas aplicable a BP, los procedimientos de transformación que proponemos y/o que utilizamos son básicamente algoritmos, a veces descritos explícitamente o a través de definiciones que describen su aplicación.

En síntesis, proponemos transformaciones de modelos (de sistemas) [156] relacionando construcciones del modelo origen con construcciones del lenguaje destino. En particular, las construcciones del lenguaje destino corresponden a las establecidas por el dominio semántico de CSP y CSP+T, ya que éste es el dominio semántico que utilizamos para conducir la verificación que proponemos. Por otra parte, tenemos como lenguajes de origen: (1) los constructos de la lógica temporal CCTL, (2) los elementos notacionales de los diagramas de máquinas de estado con anotaciones temporales de UML (UML–TSM), y (3) los elementos notacionales de BPMN que manejan tiempo. En las siguientes secciones ahondaremos más en las respectivas transformaciones.

3.4.1 Transformación de CTL/CCTL a CSP/CSP+T

Esta transformación se hace en 2 etapas. Una que corresponde a la generación del TBA semánticamente equivalente a una fórmula CCTL, y la otra es la definición de los términos de procesos en CSP+T que corresponden al TBA obtenido previamente. La razón fundamental para realizarlo en dos pasos se debe a que se hace necesario analizar primero la fórmula CCTL a lo largo del intervalo de tiempo que ésta establece para determinar sus estados de aceptación. Tal como se ejemplificó en la sección 3.2.3.1, una fórmula CCTL es representable a través de una EK como lo es un TBA. Posteriormente, en la segunda etapa, partiendo de la estructura de datos o de la representación gráfica del TBA generado a partir de la fórmula CCTL, hacemos corresponder cada uno de los elementos notacionales del TBA a los elementos del lenguaje CSP+T. Esto sobre la base de que también podemos tener una representación de un término de proceso CSP+T a través de un TBA, tal como se ejemplificó en la sección 3.2.3.2.

Más adelante, en la secciones 3.5.1 y 3.5.2, se describen detalladamente cada una de las etapas indicadas anteriormente, las cuales pueden ser automatizadas dada su descripción a través de algoritmos.

3.4.2 Transformación de UML–TSM* (ampliado con anotaciones temporales) a CSP/CSP+T

Para obtener esta transformación, utilizamos las reglas definidas en el *Método para el Diseño de Sistemas basado en Transformación Analítica de Modelos de Tiempo Real* (MEDISTAM–RT) [21], que nos permite especificar y verificar modelos de sistemas de tiempo real hechos con diagramas de clase, de estructura de composición y UML–TSM descritos originalmente en el perfil de UML para tiempo real (UML-RT) [193]. Cabe indicar que los UML–TSM que se especifican en MEDISTAM–RT son unas máquinas de estado de UML que incluyen anotaciones temporales definidas según la notación de CSP+T [209] para representar restricciones temporales, y que llamamos UML–TSM*.

A través de la aplicación de las reglas de transformación descritas en MEDISTAM–RT, obtenemos un conjunto de términos de proceso CSP+T que corresponden a un UML–TSM*. Como resultado de la transformación de un UML–TSM* a términos de procesos CSP+T, elementos de UML–RT (cápsulas, puertos, etc.), se especifican a través de procesos secuenciales definidos sintácticamente y que especifican el comportamiento de un componente, de acuerdo a eventos temporizados (según un modelo discreto de tiempo) y secuencia de eventos.

Con la finalidad de facilitar la implementación y posterior automatización del EFVC, describimos las reglas de transformación definidas en [21] de acuerdo a la Semántica Operacional Estructurada (*Structural Operational Semantics*, SOS) introducida por Plotkin en 1981 [169, 170], la cual es usualmente usada para definir la semántica formal de lenguajes de especificación basados en cálculos de proceso y lenguajes de programación específicos [169]. La SOS es considerada composicional porque permite definir la semántica de términos de procesos complejos a partir de términos más simples. La aplicación de la plantilla:

$$evento/communicación/paso\ de\ ejecución)\frac{premisas}{conclusión}(condiciones)$$

puede ser entendida como una transformación entre dos términos sintácticos que sucede como consecuencia de una *comunicación* entre procesos concurrentes o un *paso de ejecución* o una *ocurrencia de un evento* en un proceso secuencial. Así, cada regla define las *premisas* del elemento UML–RT a ser transformado y las condiciones que deben ser satisfechas antes de la transformación del elemento en cuestión en un término de proceso sintáctico CSP+T indicado en la *conclusión* de la regla. La Tabla 3.2 muestra un ejem-

plo gráfico de la aplicación de las reglas de transformación para obtener términos de proceso CSP+T a partir de un UML–TSM*, adaptado de [21].

Brevemente, la transformación se realiza mediante la asignación de: (1) cada estado del UML–TSM* a un término proceso CSP+T, (2) cada transición a un proceso guardado CSP+T (3), cada guarda de tiempo discreto a un intervalo de activación de un evento CSP+T y, (4) dos o más transiciones de salida a dos o más procesos guardados CSP+T separados por el operador de elección externa ($\Box$). Podemos establecer que la *simulación* es la relación comportamental que existe entre un término CSP+T y un UML–TSM*. Formalmente, la Definición 3.5 establece la relación de simulación que utilizamos en nuestro trabajo.

Definición 3.5. Relación de simulación. Una relación binaria $\preceq$ sobre dos sistemas de transición arbitrarias $\mathbf{S}_1(\mathrm{S}_1, \Sigma, \{\xrightarrow{e/a} | \ e, a \in \Sigma\})$ y $\mathbf{S}_2(\mathrm{S}_2, \Sigma, \{\xrightarrow{e/a} | \ e, a \in \Sigma\})$ es una simulacion si $P \preceq Q$ implica que, para cada $e, a \in \Sigma$:

$$\forall\, P \xrightarrow{e/a} P',\ \exists\, Q' \mid Q \xrightarrow{e/a} Q' \wedge P' \preceq Q'.$$

donde S_1 y S_2 son conjuntos de estados, Σ es un conjunto de señales, $\{\xrightarrow{e/a} | \ e, a \in \Sigma\}$ es un conjunto de relaciones de transición etiquetadas, $P, P' \in \mathrm{S}_1$, y $Q, Q' \in \mathrm{S}_2$.

Sobre la base de la definición anterior, el comportamiento U_i (modelado por un UML–TSM) de un componente C_i es interpretado como un proceso CSP+T Pi. Así, la relación $Pi \preceq U_i$ denota que Pi cumple con el modelo U_i, donde $\preceq$ representa que Pi *simula* U_i, lo que significa que cualquier comportamiento de U_i puede corresponderse con un comportamiento de Pi (pero no viceversa).

3.4.3 Transformación de elementos BPMN que manejan tiempo a Procesos Secuenciales de Comunicación + Tiempo

Nuestra propuesta toma como punto de partida la semántica para los elementos notacionales BPMN presentada en [218] y la combina con algunos de los operadores de CSP+T. En concreto, tomamos el operador de captura de tiempo ($\bowtie$) y el intervalo de tiempo de activación $\mathrm{I}(T, t).a$ (o $[t, t+T].a$) para especificar los tiempos de respuesta de los elementos notacionales de BPMN que manejan tiempo. Así, controlamos (1) el tiempo de duración de las actividades, de acuerdo con los tiempos máximo en los cuales deben ejecutarse, (2) los tiempos máximo de retraso de los eventos de inicio temporizado e intermedio temporizado, y (3) la sincronización de flujos de men-

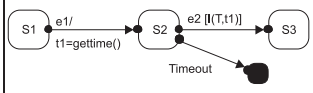
<table>
<tr><th>UML–TSM</th><th>Descripción</th></tr>
<tr><td> 
</td><td>El estado $S1$ antecede al estado $S2$ y son alcanzados cuando los eventos e_1 y e_2 ocurren, respectivamente. Sin embargo, para alcanzar el estado $S2$ el evento e_2 (evento restringido) debe ocurrir dentro del intervalo de tiempo $[t1, t1 + T]$ (intervalo de tiempo de activación), donde t_1 es la variable marcadora del evento e_1 (evento marcador). Si el evento restringido e_2 no ocurre dentro del intervalo de tiempo $[t1, t1 + T]$, es decir, el intervalo de tiempo de activación se agota, entonces se alcanza un pseudo estado Timeout. $t_1, T \in \mathbb{N}^*$ (es decir, los números naturales sin el cero).</td></tr>
<tr><td colspan="2">CSP+T SOS</td></tr>
<tr><td colspan="2">1. ocurrencia de e_1) $\frac{S1=e_1 \bowtie t_1 \rightarrow S2}{t_1=s(e_1);\ S2} \begin{pmatrix} S1, S2 \in states; \\ s(e_1) \end{pmatrix}$
2.
ocurrencia de e_2) $\frac{S2=\mathrm{I}(T,t_1).e_2 \rightarrow S3}{s(e_2);\ S3} \begin{pmatrix} s(e_2) \in [t_1, t_1 + T]; \\ S2, S3 \in states \end{pmatrix}$
O
$\mathrm{I}(T, t_1)$ timeout) $\frac{S2=\mathrm{I}(T,t_1) \rightarrow Timeout \rightarrow SKIP}{s(\tau);\ Timeout \rightarrow SKIP} \begin{pmatrix} s(\tau) < t_1 + T;\ S2 \in estados; \\ Timeout \in pseudo\ estados \end{pmatrix}$
paso de ejecución $Timeout$) $\frac{Timeout \rightarrow SKIP}{s(\tau);\ SKIP} \begin{pmatrix} s(\tau) = t_1 + T; \\ Timeout \in pseudo\ estados \end{pmatrix}$</td></tr>
</table>

Tabla 3.2 Ejemplo de SOS de una regla de transformación de UML–TSM* a términos de proceso CSP+T.

sajes entre actividades de participantes que colaboran en la realización de un BP. De esta manera se obtiene una sintaxis y una semántica más precisa tanto para los diagramas locales que representan participantes individuales como para el diagrama global que representa la colaboración de negocios requerida por el BP representado a través de un BPD.

Para definir con precisión la transformación, usamos el operador de captura de tiempo ($\bowtie$) para *inicializar* (asignarle un valor inicial) a la variable marcadora v cuando ocurra el evento marcador a que invoca al elemento notacional BPMN; es decir, $a \bowtie v$. Esta variable marcadora es utilizada como 'un cronómetro local' relativo al instante de inicio de los elementos de BPMN que manejan tiempo, que permite precisar la semántica de la restricción de la ocurrencia de los eventos que dependen de la ocurrencia del evento marcador o instante de inicio del elemento notacional BPMN. En particular, la incorporación de estos operadores nos permiten precisar el tiempo mínimo y el tiempo máximo de duración de los elementos de tiempo de BPMN. De esta manera, logramos precisar: (1) el tiempo de inicio de elementos notacionales BPMN que están a continuación de los elementos de BPMN que manejan tiempo, según el flujo de secuencia, con la finalidad de garantizar que se cumplan los tiempos requeridos por el BP, y (2) la sincronización de las actividades involucradas en flujos de mensajes, con la finalidad de garantizar la colaboración requerida entre los distintos participantes que realizan un BP.

La Tabla 3.3 muestra gráficamente las reglas de transformación que permiten obtener los términos CSP+T a partir de entidades de modelado incluidas en BPMN que manejan tiempo. Denotamos como ε_x a los eventos de invocación de las entidades estándar de modelado BPMN [158], $Sx.ran.min$ y $Sx.ran.max$ como el tiempo mínimo y máximo, respectivamente, de duración de la actividad $bpmn\langle\langle Sx\rangle\rangle$; y T como el tiempo de demora definido por el evento de inicio temporizado $stime\langle\langle T\rangle\rangle$ y el evento intermedio temporizado $itime\langle\langle T\rangle\rangle$.

En resumen, la transformación se realiza mediante la asignación de: (1) cada una de las entidades de modelado BPMN que manejan tiempo a un término prefijo de CSP+T, (2) cada tiempo de duración discreto a un intervalo de habilitación de un evento, y (3) se utiliza la elección externa para especificar las alternativas que el entorno de los procesos, cuya composición representa al BP (tal como mostramos en las aplicaciones descritas en el capítulo 6), puede ejercer sobre los términos de proceso CSP+T para garantizar su terminación cuando acabe la ejecución del modelo del BP.

3.5 Tecnología para el sistema de verificación

En esta sección se presentan los algoritmos y procedimientos que se han propuesto para dotar a nuestro enfoque de un soporte automatizado para la verificación de BP, y que se basa en utilizar el análisis temporal que se presentó en la sección 3.3. Los desarrollos presentados en esta sección sientan las bases para la implementación de herramientas de software y se consideran en un punto de desarrollo para próximos trabajos continuación de esta tesis doctoral.

3.5.1 Algoritmo de generación de Autómatas de Büchi a partir de fórmulas CCTL

Este algoritmo se enmarca dentro de una estrategia de verificación de propiedades CCTL que se basa en TBAs. Para alcanzar este objetivo, el algoritmo plantea un procedimiento iterativo de construir un TBA determinista semánticamente equivalente a una fórmula CCTL. Tomando en cuenta el estado en el cual se encuentra el autómata junto con el siguiente operador y su tiempo de aceptación, se obtiene deterministicamente el próximo cambio de estado. Este objetivo es gradualmente alcanzado siguiendo un proceso que comienza analizando la fórmula temporal CCTL y luego construye el grafo

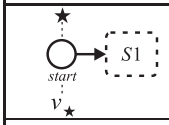
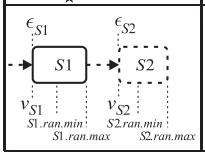
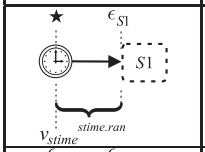
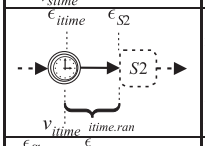
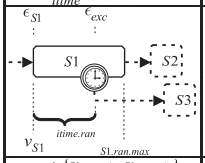
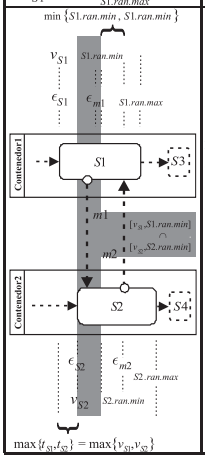
Elemento BPMN	Descripción	Proceso CSP+T
	El evento de inicio simple (*start*) corresponde al evento especial de instanciación $\star$ de CSP+T y la variable marcadora $v_\star$ es usada para almacenar el instante de ocurrencia del evento $\star$.	$P(start) = (\star \bowtie v_\star \rightarrow SKIP \fatsemi P(S1)) \Box (\varepsilon_{end} \rightarrow SKIP)$
	La actividad $S1$ antecede a la actividad $S2$. La actividad $S1$ comienza cuando ocurre el evento ε_{S1} y la invocación de la actividad $S2$ (lo cual corresponde a la ocurrencia del evento ε_{S2}) debe ocurrir dentro del intervalo de tiempo $[S1.ran.min, S1.ran.max]$.	$P(S1) = (\varepsilon_{S1} \bowtie v_{S1} \rightarrow SKIP \fatsemi$ $\mathrm{I}(S1.ran.max - S1.ran.min,$ $v_{S1} + S1.ran.min).\varepsilon_{S2}$ $\rightarrow SKIP \fatsemi P(S2))$ $\Box(\varepsilon_{end} \rightarrow SKIP)$
	El inicio temporizado (*stime*) establece que la actividad $S1$ debe comenzar (lo cual corresponde a la ocurrencia del evento ε_{S1}) *stime.ran* unidades de tiempo después de la ocurrencia del evento especial de instanciación $\star$ de CSP+T.	$P(stime) = (\star \bowtie v_{stime} \rightarrow SKIP \fatsemi$ $\mathrm{I}(stime.ran, v_{stime}) \rightarrow SKIP \fatsemi$ $\varepsilon_{S1} \rightarrow SKIP \fatsemi P(S1))$ $\Box(\varepsilon_{end} \rightarrow SKIP)$
	De acuerdo con el intermedio temporizado (*itime*), la actividad $S2$ debe comenzar (lo cual corresponde a la ocurrencia del evento ε_{S2}) *itime.ran* unidades de tiempo después de la ocurrencia del evento ε_{itime}.	$P(itime) = (\varepsilon_{itime} \bowtie v_{itime} \rightarrow SKIP \fatsemi$ $\mathrm{I}(itime.ran, v_{itime}) \rightarrow SKIP \fatsemi$ $\varepsilon_{S2} \rightarrow SKIP \fatsemi P(S2))$ $\Box(\varepsilon_{end} \rightarrow SKIP)$
	La ejecución de la actividad $S1$ puede ser interrumpida (lo cual corresponde a la ocurrencia del evento ε_{exc}) *itime.ran* unidades de tiempo después de su inicio (lo cual corresponde a la ocurrencia del evento ε_{S1}), es decir, dentro del intervalo de tiempo $[v_{S1}, itime.ran]$).	$P(S1) = (\varepsilon_{S1} \bowtie v_{S1} \rightarrow SKIP \fatsemi$ $\mathrm{I}(S1.ran.max - S1.ran.min,$ $v_{S1} + S1.ran.min).\varepsilon_{S2} \rightarrow$ $(SKIP \bigtriangleup \mathrm{I}(itime.ran, v_{S1}) \rightarrow SKIP \fatsemi$ $\varepsilon_{exc} \rightarrow SKIP \fatsemi P(S3))$ $\Box(\varepsilon_{end} \rightarrow SKIP)$
	Para los flujos de mensaje, la actividad $S1$ envía el mensaje $m1$, el cual es recibido por la actividad $S2$. La actividad $S2$ entonces responde enviando el mensaje $m2$. Esto sólo es posible dentro del intervalo de tiempo representado por el área sombreada, el cual corresponde con la intersección de los intervalos de tiempo $[v_{S1}, v_{S1} + S1.ran.min]$ y $[v_{S2}, v_{S2} + S2.ran.min]$, donde $MI = \min\{S1.ran.min, S2.ran.min\}$ y $MA = \max\{v_{S1}, v_{S2}\}$.	$P(S1) = (\varepsilon_{S1} \bowtie v_{S1} \rightarrow SKIP \fatsemi$ $\mathrm{I}(MI, MA).\varepsilon_{m1}!x \rightarrow SKIP \fatsemi$ $\mathrm{I}(MI, MA).\varepsilon_{m2}?y \rightarrow SKIP \fatsemi$ $\mathrm{I}(S1.ran.max - S1.ran.min, v_{S1} +$ $S1.ran.min).\varepsilon_{S3} \rightarrow SKIP \fatsemi P(S3))$ $\Box(\varepsilon_{end.1} \rightarrow SKIP)$ $P(S2) = (\varepsilon_{S2} \bowtie v_{S2} \rightarrow SKIP \fatsemi$ $\mathrm{I}(MI, MA).\varepsilon_{m1}?x \rightarrow SKIP \fatsemi$ $\mathrm{I}(MI, MA).\varepsilon_{m2}!y \rightarrow SKIP \fatsemi$ $\mathrm{I}(S2.ran.max - S2.ran.min, v_{S2} +$ $S2.ran.min).\varepsilon_{S4} \rightarrow SKIP \fatsemi P(S4))$ $\Box(\varepsilon_{end.2} \rightarrow SKIP)$

Tabla 3.3 Reglas de correspondencia entre BPMN y CSP+T.

que permite la generación del TBA semánticamente equivalente a la formula CCTL.

A continuación, en las sub–secciones 3.5.1.1 a 3.5.1.4, se detallarán los distintos sub–algoritmos que conforman el algoritmo completo de generación del TBA que corrrespnde a una fórmula CCTL.

3.5.1.1 Sub–algoritmo de construcción

El sub–algoritmo que se muestra en la Fig. 3.5, el cual es una adaptación de los algoritmos descritos en [76, 96], guía la generación del TBA [95].

```
1  record Nodes_Set=[Name:string,Predecessor:set of string,
2    Unprocessed:set of formula,Processed:set of formula,
3    Next:set of formula,Accept:integer];
4
5  record AcceptingCond=[Key:formula,SubformulasSet:set of formula];
6
7  function create_graph(ϑ)
8    return(expand([Name⇐new_name(),Predecessor⇐ {init},
9      Unprocessed⇐ {ϑ},Processed⇐ {∅},Next⇐ {∅},Accept⇐ ∅],{∅}))
10   return(accepting_states(Nodes_Set,ϑ))
11   return(draw_tba(Nodes_Set,Graph_tool))
12  end create_graph;
```

Fig. 3.5 Sub–algoritmo de generación del TBA.

Para el funcionamiento del sub–algoritmo, se definen dos estructuras de datos:

1. La estructura de datos `Nodes_Set` (líneas 1–3), usada para guardar la información de los nodos a lo largo del proceso está compuesta por 6 campos:
   a. su *Name*;
   b. *Predecessor*: contiene los nombres de los nodos que tienen un arco dirigido a él;
   c. *Unprocessed*: es el conjunto de fórmulas que deben mantenerse y que no han sido procesadas todavía;
   d. *Processed*: contiene las fórmulas que ya han sido analizadas;
   e. *Next*: es el conjunto de fórmulas que deben mantenerse en todos sus sucesores inmediatos; y
   f. *Accept*[2]: indica si el nodo representa un estado de aceptación.

[2] Este campo es usado por el sub–algoritmo descrito en la sección 3.5.1.4.

2. La estructura de datos `AcceptingCond_Set` (línea 4) usada para guardar la información de las condiciones de aceptación a través del proceso está compuesta por dos campos:

   a. *Key*, contiene la fórmula común a todas las sub–fórmulas CCTL que son parte del conjunto de aceptación; y
   b. *SubformulasSet*, contiene las subfórmulas del conjunto *Subformulas-Set* que corresponden a la fórmula contenida en el campo Key.

La función create_graph() (codificada en las líneas 7–12), comienza analizando la formula temporal CCTL y luego construye el grafo (function expand() invocada en la línea 8), la cual, antes de todo, genera el nodo inicial de la formula CCTL ϑ (líneas 8–9). Este nodo tiene solo un arco de entrada (*Predecessor*), etiquetado "init", para denotar que es el nodo inicial. Por otra parte, sólo tiene una única obligación inicial en el campo *Unprocessed* llamada ϑ, y los campos *Processed* y *Next* quedan inicialmente vacíos ($\{\varnothing\}$). Así, la estructura de datos `Nodes_Set` está inicializada.

Después, usando la información almacenada en la estructura de datos `Nodes_Set` del grafo de la primera fase, por invocación de la función accepting_states() (línea 10), actualizamos esta estructura de datos con la información necesaria para obtener la definición del TBA en la sección 3.2.3. Finalmente, usando la estructura de datos `Nodes_Set` actualizada podemos dibujar la representación gráfica (función draw_tba() invocada en la línea 11) del TBA semánticamente equivalente a la fórmula CCTL. En la siguiente sección el procedimiento es explicado en su totalidad.

3.5.1.2 Análisis de fórmulas CCTL

Para llevar a cabo el análisis de las fórmulas CCTL, se deben cumplir un conjunto de reglas. La aplicación de estas reglas permite *descomponer* la fórmula en los diferentes operadores y sub–fórmulas que ella incluye. Por ende, los estados de aceptación del autómata pueden ser obtenidos, los cuales especifican la propiedad que se espera cumpla el BP. Estas reglas son conocidas como el conjunto reducción (en inglés, *reductor set*) (ver Definición 3.7), ya que éstas constituyen un conjunto de condiciones para llevar a cabo la reducción de fórmulas complejas de CCTL en sub–fórmulas CCTL simples (ver Definición 3.6), las cuales son equivalentes a la fórmula original.

Definición 3.6. Relación de reducción. Entre dos formulas CCTL, φ y φ', hay una *relación de reducción* con respecto a la fórmula $\rho \in \mathbf{red}(\varphi)$, denotado como $\varphi \ {}_{\rho}\ succ\ \varphi'$, la cual se lee "$\varphi$ es ρ-reducible a φ'" o "φ' es un ρ-reductor de φ", si y sólo si una de las reglas mostradas en la Tabla 3.4 es aplicable,

donde φ es la fórmula original, ρ_1 es el conjunto reductor de la fórmula φ para obtener la fórmula φ' que debe satisfacer en el futuro, y ρ_2 es el conjunto reductor de la fórmula φ que hace que sea verdad en la instancia que ocurre. φ' es la formula reducida por la aplicación del ρ-reductor.

φ $\eta^{(*)}$	ρ_1 **New1**$(\eta)^{(*)}$	φ' **Next2**$(\eta)^{(*)}$	ρ_2 **New2**$(\eta)^{(*)}$
$f\ \mathsf{U}_{[a,b]}\ g$	$\{g\}$	$f\ \mathsf{U}_{[a+1,b]}\ g$	$\{f\}$
$f\ \mathsf{U}_{[b,b]}\ g$	$\{\varnothing\}$	$\varnothing$	$\{g\}$
$f\ \mathsf{R}_{[a,b]}\ g$	$\{f,g\}$	$f\ \mathsf{R}_{[a+1,b]}\ g$	$\{g\}$
$f\ \mathsf{R}_{[b,b]}\ g$	$\{\varnothing\}$	$\varnothing$	$\{g\}$
$f\ \vee\ g$	$\{g\}$	$\varnothing$	$\{f\}$
(*) Estos nombres de campos son usados por el algoritmo descrito en la sección 3.5.1.3.			

Tabla 3.4 Reglas de reducción.

Definición 3.7. Conjunto reducción. El *conjunto reducción* **red**(φ) de una fórmula CCTL φ, las cuales no contienen la fórmula inicial φ, tal que las siguientes reglas son aplicables:

- Si $\varphi = \mathsf{EX}_{[a]}\varphi_1$, $\mathsf{EF}_{[a,b]}\varphi_1$, $\mathsf{EG}_{[a,b]}\varphi_1$ or $\mathsf{E}(\varphi_1\mathsf{U}_{[a,b]}\varphi_2)$, entonces $\varphi_1, \varphi_2 \in$ **red**(φ)
- Si $\varphi = \neg\varphi_1$, $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $\varphi_1 \rightarrow \varphi_2$ or $\varphi_1 \leftrightarrow \varphi_2$, entonces φ_1 y $\varphi_2 \subseteq$ **red**(φ)
- Si φ es de alguna otra forma, entonces **red**$(\varphi) = \{\varnothing\}$

Por la combinación de las definiciones anteriores, mediante un proceso que simplifica la verificación de propiedades, como las expresadas en la fórmula CCTL inicial, es posible construir un grafo definiendo los estados y transiciones que representan la formula CCTL. Estas definiciones permiten garantizar que cualquier condición expresada por un reductor ρ es mantenida. Formalmente, cuando ρ es válido en una instancia de tiempo i de una EI (ver Definición 2.5) $\mathbf{I}$ $(\mathbf{I}, g_i) \models \rho$, entonces $(\mathbf{I}, g_i) \models \varphi$ si y sólo si $(\mathbf{I}, g_i) \models \varphi'$.

3.5.1.3 Sub–algoritmo de construcción del grafo

El sub–algoritmo mostrado en la Figura 3.6, el cual es una adaptación de los algoritmos descritos en [76, 96], permite la generación de un grafo o un Sistema de Transición Etiquetado (*Labelling Transition System*, LTS), donde los nodos del grafo son etiquetados por el conjunto de sub–fórmulas CCTL. Estas sub–fórmulas son obtenidas por fórmulas de descomposición de acuerdo a la estructura Boleana de la fórmula, por expansión de los operadores temporales, con el fin de separar los que son inmediatamente ver-

daderos de los que están satisfechos a partir del siguiente estado. Este algoritmo está basado en los presentados en [76, 95, 96], usa la estrategia de Primera–Búsqueda–Profunda (*Depth–First–Search*, DFS) [76], y se orienta por las formulas de lógica temporal interpretadas de acuerdo a una lógica de intervalo.

```
1  function expand(Node,Nodes_Set)
2    if Unprocessed(Node)={∅} then
3      if ∃ND∈Nodes_Set with Processed(ND)=Processed(Node) and
4        Next(ND)=Next(Node) then
5        Predecessor(ND):=Predecessor(ND)∪Predecessor(Node);
6        return(Nodes_Set);
7      else return(expand([Name⇐new_name(),
8        Predecessor⇐{Name(Node)},Unprocessed⇐Next(Node),
9        Processed⇐ {∅},Next⇐ {∅},
10       Accept⇐ ∅],{Node}∪Nodes_Set))
11   else
12     let η ∈Unprocessed;
13     Unprocessed(Node):=Unprocessed(Node)\{η};
14     case η of
15     η = P_n or ¬P_n or True or False or φ U_[b,b]ψ or φ R_[b,b]ψ =>
16       if η = False or Neg(η) ∈Processed(Node)
17         then return(Nodes_Set)
18       else Processed(Node):=Processed(Node)∪{η};
19         return(expand(Node,Nodes_Set));
20     η = φ U_[a,b]ψ or φ R_[a,b]ψ or φ∨ψ =>
21       Node1:=[Name⇐new_name(),Predecessor⇐Predecessor(Node),
22         Unprocessed⇐Unprocessed(Node)∪({New1(η)}\Processed(Node)),
23         Processed⇐Processed(Node)∪{η}, Next⇐Next(Node),
24         Accept⇐ ∅];
25       Node2:=[Name⇐Name(Node),Predecessor⇐Predecessor(Node),
26         Unprocessed⇐Unprocessed(Node)∪({New2(η)}\Processed(Node)),
27         Processed⇐Processed(Node)∪{η},
28         Next⇐Next(Node)∪{Next2(η)},Accept⇐ ∅];
29       return(expand(Node2,expand(Node1,Nodes_Set)));
30      η = φ∧ψ =>
31       return(expand([Name⇐Name(Node),Predecessor⇐Predecessor(Node),
32         Unprocessed⇐Unprocessed(Node)∪({φ,ψ}\Processed(Node)),
33         Processed⇐Processed(Node)∪{η},Next⇐Next(Node)],
34         Accept⇐ ∅],Nodes_Set))
35 end expand;
```

Fig. 3.6 Sub–algoritmo de generación del grafo.

El sub–algoritmo comienza a partir del nodo descrito en la sección 3.5.1.1. Iterativamente busca y analiza cada uno de los nodos que muestra obligaciones por cumplir en el campo *Unprocessed* (líneas 12–34). Por el contrario, si el nodo muestra *Unprocessed*$=\{\varnothing\}$, entonces el nodo se une al `Node_Set` (líneas 2–10). En el caso de ya estar como un nodo procesado

en el `Node_Set` con la misma información en los campos *Predecessor* y *Next* del nodo que se incluye, entonces se actualiza el `Node_Set` añadiendo el conjunto de arcos de entrada del nuevo nodo al campo *Predecessor* (líneas 3–6).

Si el nodo no existe en el `Node_Set` (líneas 7-10), el nodo es agregado a la lista y un nuevo nodo es construido por su sucesor, de acuerdo a lo siguiente:

1. el campo *Predecessor* del nodo sucesor es inicializado con el nombre del nodo el cual fue agregado al `Node_Set`;
2. el campo *Unprocessed* es inicializado con el contenido del campo *Next* del último; y
3. los campos *Processed* y *Next* del nodo sucesor están inicialmente vacíos ($\{\varnothing\}$).

Cuando un nodo es procesado, la fórmula η es removida del campo *Unprocessed* (línea 13). En caso que η sea una proposición o su negación, entonces $\neg\eta$ ya se encuentra en el campo *Processed* ($\neg\neg\eta$ es identificada con η) y el nodo actual es descartado (líneas 16–17), ya que contiene una contradicción. En caso contrario, η es agregado al campo *Processed* (líneas 18–19). Cuando η no es una proposición, el nodo es dividido en dos nodos o se convierte en un nuevo nodo (líneas 20–34); en cualquier caso, nuevas fórmulas son agregadas a los campos *Unprocessed* y *Next* dependiendo de:

$\eta = \varphi \wedge \psi$. Ambas fórmulas φ y ψ son agregadas al campo *Unprocessed* como verdaderos y ambos son necesarios para que η se mantenga.

$\eta = \varphi \vee \psi$. El nodo es dividido, agregando φ al campo *Unprocessed* de una copia, y ψ a la otra copia. Estos nodos corresponden a dos maneras en la cual η se puede mantener.

$\eta = \varphi \mathsf{U}_{[a,b]} \psi$. El nodo es dividido. Para la primera copia, ψ es agregada al campo *Next*. Para la otra copia, φ es agregada al campo *Unprocessed* y $\varphi \ \mathsf{U}_{[a+1,b]} \ \psi$, al campo *Next*. Esta división es explicada observando que $\varphi \ \mathsf{U}_{[a,b]} \ \psi$ es equivalente a ($a \in \mathbb{N}$ y $b \in \mathbb{N} \cup \{\infty\}$ son los extremos del intervalo):

$$\varphi \mathsf{U}_{[a,b]} \psi :\equiv \begin{cases} \varphi \vee \mathsf{X}[\varphi \mathsf{U}_{[a+1,b]} \psi] & \text{si } a < b \\ \psi \ \wedge \neg\varphi & \text{si } a = b \end{cases} \tag{3.14}$$

$\eta = \varphi \mathsf{R}_{[a,b]} \psi$. El nodo es dividido, agregando ψ al campo *Unprocessed* de ambas copias, φ al campo *Unprocessed* de la primera copia, y $\varphi \mathsf{R}_{[a+1,b]}$ ψ, al campo *Next* de la segunda copia. Esta división es explicada observando que $\varphi \mathsf{R}_{[a,b]} \psi$ es equivalente a ($a \in \mathbb{N}$ y $b \in \mathbb{N} \cup \{\infty\}$ son los extremos del intervalo):

$$\varphi \mathsf{R}_{[a,b]} \psi :\equiv \begin{cases} \psi \wedge \mathsf{X}[\varphi \mathsf{R}_{[a+1,b]} \psi] & \text{si } a < b \\ \psi \ \wedge \ \varphi & \text{si } a = b \end{cases} \tag{3.15}$$

El sub–algoritmo en pseudo-código es descrito en la Fig. 3.6. Usa la función new_name() para generar el nombre de cada nodo, la función **Neg**, la cual se define **Neg**(P_n) y **Neg**($\neg P_n$) y las funciones **New1**(η), **New2**(η), y **Next2**(η), las cuales corresponden a las celdas ρ_1, ρ_2 and φ', respectivamente, de la Tabla 3.4.

3.5.1.4 Sub–algoritmo de generación del Autómata de Büchi

Ahora se muestra como derivar, desde la estructura de datos obtenida por el la ejecución del sub–algoritmo de construcción del grafo, un TBA que se ajusta a la definición descrita en la sección 3.2.3. El sub–algoritmo de la sección 3.5.1.3 nos permite construir el grafo o LTS que corresponde a una fórmula CCTL, pero no el TBA equivalente a ella. Por lo tanto, necesitamos determinar los estados de aceptación y de no aceptación del autómata, los cuales corresponden con sus ejecuciones de aceptación. Para lograr este objetivo, seguimos el sub–algoritmo mostrado en la Fig. 3.7, el cual es una adaptación del algoritmo descrito en [96].

```
1  function accepting_states(Nodes_Set,ϑ)
2    AcceptingConds_Set⇐[Key⇐ ϑ,SubformulasSet⇐ {∅}];
3    for each Node∈Nodes_Set do
4      if χ∈Next(Node) is a eventuality then
5        SubformulasSet:=SubformulasSet∪{χ}];
6    return(AcceptingConds_Set);
7    for each Node∈Nodes_Set do
8      if Next(Node)∈SubformulasSet then
9        Accept(Node):=0
10     else
11       Accept(Node):=1;
12  end accepting_states;
```

Fig. 3.7 Sub–algoritmo para la determinación de estados de aceptación y de no aceptación.

El sub–algoritmo en pseudo–código descrito en la Fig. 3.7 busca el campo *Next* en la estructura de datos `Nodes_Set` (líneas 3–6) del grafo construido a partir de ϑ, para almacenar las sub–fórmulas CCTL que representan una *eventualidad*[3] (líneas 4-5) en el campo *SubformulasSet* de la estructura de datos `AcceptingConds_Set`. El conjunto formado por estas sub–fórmulas define las eventualidades que afectan la satisfacción de la fórmula CCTL; es decir, son las *condiciones de aceptación* de la fórmula.

[3] Aquellas sub–fórmulas que pueden posponer su satisfacción de la fórmula CCTL (en este caso, ϑ) a lo largo del intervalo de tiempo.

Finalmente, el sub–algoritmo determina los estados *aceptados* y *no aceptados* que pertenecen al TBA. El sub–algoritmo verifica para cada nodo (líneas 7–11) si el campo *Next* contiene alguna condición de aceptación (es decir, una eventualidad) determinada previamente. En caso de que el campo *Next* contenga alguna condición de aceptación (línea 8) esto significa que el estado representado por el nodo *no corresponde a un estado de aceptación* debido a que este nodo tiene eventualidades que no se satisfacen aun o alguna condición de aceptación no es inmediatamente satisfecha en este estado. De lo contrario, el nodo representa un estado de aceptación debido a que no tiene eventualidades para ser satisfecho. El sub–algoritmo asigna en valor '0' (línea 9) en el campo *Accept* de cada nodo registrado en la estructura de datos `Nodes_Set` en el caso de que el nodo no representa un estado de aceptación. Si el nodo representa un estado de aceptación se asigna el valor '1' (línea 11) en el campo *Accept*.

3.5.1.5 Visualización de la representación gráfica del Autómata de Büchi

Con la información guardada en la actualización de la estructura de datos `Node_Set` es posible visualizar la representación gráfica del TBA. Esto es una fase opcional. El sub–algoritmo mostrado en la Fig. 3.8 resume esta tarea, donde podemos usar cualquier software de visualización gráfica para implementarlo.

```
1  function draw_tba(Nodes_Set,Graph_tool))
2    TBARepresentation:=draw the TBA stored in Nodes_Set;
3    display TBARepresentation;
4  end draw_tba;
```

Fig. 3.8 Sub–algoritmo de visualización del TBA.

En el Apéndice B se presenta la ejecución del algoritmo aquí descrito a través de un ejemplo. La formula CCTL usada para mostrar la ejecución del algoritmo es $\vartheta = \mathsf{E}(\mathsf{F}_{[1,3]}\varphi)$, la cual expresa la verdad de la proposición atómica φ en el futuro, dentro del intervalo $[1,3]$.

3.5.2 *Definición para la construcción de términos de Procesos Secuenciales de Comunicación + Tiempo a partir de Autómatas de Büchi*

La Definición 3.8, que se presenta a continuación, establece las reglas que permiten obtener los términos de procesos CSP+T partiendo de una estructura de datos como `Nodes_Set` (ver sección 3.5.1) o de la representación gráfica de un TBA (ver Apéndice B).

Definición 3.8. Transformación de un TBA a un proceso CSP+T. La transformación CSP+T(**A**) de un TBA **A** en un proceso CSP+T es la siguiente:

- Cada $s \in S$ es mapeado a un proceso $P(s)$ de CSP+T.
- Para cada $s \in I$ se agrega la definición del proceso CSP+T: $NAME = t_0.\star \rightarrow P(s)$.
- Para cada estado de no aceptación $s \in S \backslash F$ y para todos los arcos de salida $(e, s, s', i) \in R$, donde i es una restricción temporal en forma de intervalo de tiempo $[a, b]$, se agrega el proceso CSP+T: $P(s) = \mathrm{I}(b - a, a).e \rightarrow P(s')$
- Para cada estado de no aceptación[4] $s \in S \backslash F$, donde $\{(e_1, s, s_1, i_1), \ldots, (e_n, s, s_n, i_n)\} \subseteq R$ son todos los arcos de salida de s, y $i_1, \ldots, i_n$ son restricciones temporales en la forma de intervalos de tiempo $[a_1, b_1], \ldots, [a_n, b_n]$, respectivamente, se agrega el proceso CSP+T: $P(s) = \mathrm{I}(b_1 - a_1, a_1).e_1 \rightarrow P(s_1) \,\Box \ldots \Box\, \mathrm{I}(b_n - a_n, a_n).e_n \rightarrow P(s_n)$
- Para cada estado de aceptación $s \in F$, se agrega el proceso CSP+T: $P(s) = SKIP \fatsemi NAME$

En el Apéndice C se presenta la aplicación de las reglas que conforman la Definición 3.8 a través de un ejemplo. El TBA usado para mostrar la aplicación es el obtenido por la ejecución del algoritmo descrito en la sección 3.5.1 (ver Apéndice B).

3.5.3 *Semántica Operacional Estructurada de los términos de proceso CSP+T derivados de BPMN*

Con el objeto de facilitar la implementación de las reglas de transformación que nos permiten obtener los términos de procesos CSP+T a partir de los elementos notacionales de BPMN que manejan tiempo, compilados en la Fig. 3.9, a continuación presentamos su especificación de acuerdo a la plantilla de SOS ya presentada en la sección 3.4.2. El uso de SOS nos sirvió de

[4] ver sección 3.2.3 para mayores detalles.

base para el desarrollo de la herramienta que llamamos BTransformer y que se presenta en el capítulo 5.

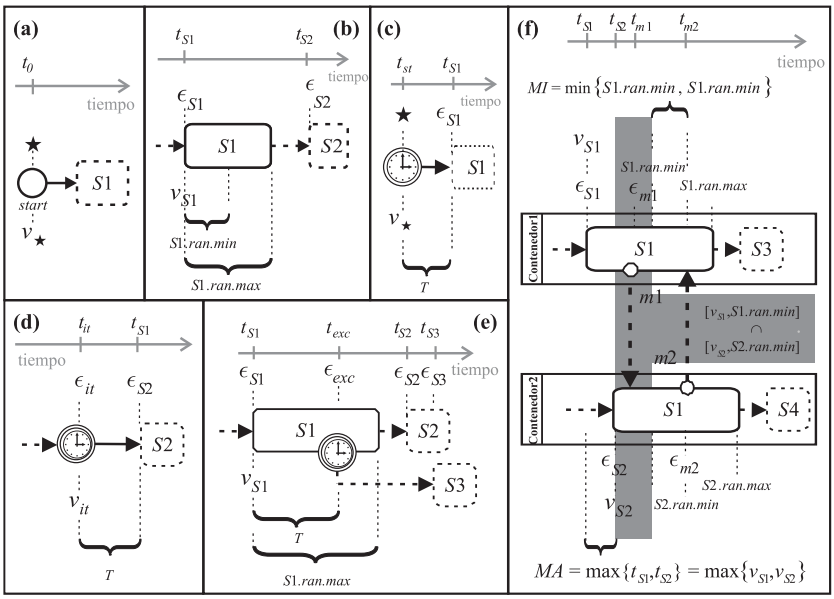


Fig. 3.9 Compilación de los elementos notacionales de BPMN que manejan tiempo.

En la Fig. 3.9 (a) el *Evento de inicio simple* (*start*) de BPMN es mostrado gráficamente, el cual representa la instanciación necesaria de un BP en particular antes de iniciarse su ejecución. En CSP+T, esta especificación es obtenida mediante el uso del *evento de instanciación* $\star$. Cuando este evento surge, su instante de ocurrencia ($s(\star)$) es almacenado en las variable marcadora $v_\star$. La regla según la estructura SOS queda como se muestra en la Fig. 3.10 (denotamos con *ENT* al conjunto de todos los tipos de entidades de modelado de BPMN definidos en [214]).

$$\text{occurrencia de } \star)\ \frac{\begin{array}{c}P(start) = (\star \bowtie v_\star \rightarrow SKIP \fatsemi P(S1)) \\ \Box(\varepsilon_{end} \rightarrow SKIP)\end{array}}{v_\star = s(\star);\ \ SKIP \fatsemi P(S1) \Box \varepsilon_{end} \rightarrow SKIP} \begin{pmatrix} start;\ s(\star); \\ S1 \in ENT \backslash \{start\} \end{pmatrix}$$

Fig. 3.10 SOS del término de proceso correspondiente al evento de inicio simple de BPMN.

Sea la *Actividad* $bpmn\langle\langle S1\rangle\rangle$ que precede a la *Actividad* $bpmn\langle\langle S2\rangle\rangle$, de acuerdo con el flujo de secuencia mostrado en la Fig. 3.9 (b). De acuerdo con la semántica que proponemos, el inicio de la ejecución de $bpmn\langle\langle S2\rangle\rangle$

(la ocurrencia del evento ε_{S2}) depende de la finalización de la actividad $bpmn\langle\langle S1\rangle\rangle$ (invocada por la ocurrencia del evento ε_{S1}), la cual debe ocurrir dentro del tiempo de duración de la actividad $bpmn\langle\langle S1\rangle\rangle$, dado por el intervalo de tiempo $A = [S1.ran.min,\ S1.ran.max]$. A su vez, la medición de los rangos $S1.ran.min$ y $S1.ran.max$ se hace a partir de la ocurrencia del evento ε_{S1}. Así, aseguramos que el evento ε_{S2} ocurrirá con puntualidad; es decir, dentro del intervalo A referido al instante t_{S1} en el cual se invocó la actividad $bpmn\langle\langle S1\rangle\rangle$, y almacenado en v_{S1}. En CSP+T el término de proceso que especifica este comportamiento se obtiene operacionalmente de acuerdo a la Fig. 3.11[5]:

$$\text{1. occurrencia de } \varepsilon_{S1})\ \frac{\begin{array}{l}P(S1) = (\varepsilon_{S1} \bowtie v_{S1} \rightarrow SKIP \fatsemi \\ \qquad \mathrm{I}(S1.ran.max - S1.ran.min, v_{S1} + S1.ran.min).\varepsilon_{S2} \\ \qquad \rightarrow SKIP \fatsemi P(S2)) \Box (\varepsilon_{end} \rightarrow SKIP)\end{array}}{\begin{array}{l}v_{S1} = s(\varepsilon_{S1});\ SKIP \fatsemi \\ \mathrm{I}(S1.ran.max - S1.ran.min, v_{S1} + S1.ran.min).\varepsilon_{S2} \\ \rightarrow SKIP \fatsemi P(S2) \Box (\varepsilon_{end} \rightarrow SKIP)\end{array}} \begin{pmatrix} bpmn\langle\langle S1\rangle\rangle;\ s(\varepsilon_{S1}); \\ S1 \in ENT \backslash \{start\}\end{pmatrix}$$

$$\text{2. occurrencia de } \varepsilon_{S2})\ \frac{\begin{array}{l}\mathrm{I}(S1.ran.max - S1.ran.min, v_{S1} + S1.ran.min).\varepsilon_{S2} \\ \rightarrow SKIP \fatsemi P(S2) \Box (\varepsilon_{end} \rightarrow SKIP)\end{array}}{s(\varepsilon_{S2});\ SKIP \fatsemi P(S2) \Box (\varepsilon_{end} \rightarrow SKIP)} \begin{pmatrix} s(\varepsilon_{S2}) \in A; \\ S2 \in ENT \backslash \{start\}\end{pmatrix}$$

Fig. 3.11 SOS del término de proceso correspondiente a una actividad de BPMN.

El *Evento de inicio temporizado* de BPMN $stime\langle\langle T\rangle\rangle$ establece que la entidad de modelado $bpmn\langle\langle S1\rangle\rangle$ a la cual precede en el *flujo de secuencia*, debe comenzar T ($T \in \mathbb{N}^*$) unidades de tiempo después de la ocurrencia del evento de instanciación $\star$ de CSP+T. De acuerdo al esquema mostrado en la Fig. 3.9 (c), el término de proceso CSP+T que especifica ese comportamiento sigue la estructura descrita en la Fig. 3.12:

$$\text{1. occurrencia de } \star)\ \frac{\begin{array}{l}P(stime) = (\star \bowtie v_\star \rightarrow SKIP \fatsemi \mathrm{I}(T, v_\star) \rightarrow SKIP \fatsemi \\ \qquad \varepsilon_{S1} \rightarrow SKIP \fatsemi P(S1)) \Box (\varepsilon_{end} \rightarrow SKIP)\end{array}}{\begin{array}{l}v_\star = s(\star);\ SKIP \fatsemi \mathrm{I}(T, v_\star) \rightarrow SKIP \fatsemi \\ \varepsilon_{S1} \rightarrow SKIP \fatsemi P(S1) \Box (\varepsilon_{end} \rightarrow SKIP)\end{array}} (\ stime\langle\langle T\rangle\rangle;\ s(\star))$$

$$\text{2. } \mathrm{I}(T, v_\star) \text{ timeout})\ \frac{\begin{array}{c}\mathrm{I}(T, v_\star) \rightarrow SKIP \fatsemi \varepsilon_{S1} \rightarrow SKIP \fatsemi \\ P(S1) \Box (\varepsilon_{end} \rightarrow SKIP)\end{array}}{s(\tau);\ SKIP \fatsemi \varepsilon_{S1} \rightarrow SKIP \fatsemi P(S1) \Box (\varepsilon_{end} \rightarrow SKIP)} (\ s(\tau) < v_\star + T\ )$$

$$\text{3. ocurrencia de } \varepsilon_{S1})\ \frac{\varepsilon_{S1} \rightarrow SKIP \fatsemi P(S1) \Box (\varepsilon_{end} \rightarrow SKIP)}{s(\varepsilon_{S1});\ SKIP \fatsemi P(S1) \Box (\varepsilon_{end} \rightarrow SKIP)} \begin{pmatrix} s(\varepsilon_{S1}) = v_\star + T; \\ S1 \in ENT \backslash \{start\}\end{pmatrix}$$

Fig. 3.12 SOS del término de proceso correspondiente al evento de inicio temporizado de BPMN.

[5] $\{Sx.ran.min, Sx.ran.max\} \in \mathbb{N}^*$; es decir, el conjunto de los números naturales sin el cero.

El *Evento intermedio temporizado* $itime\langle\langle T\rangle\rangle$ especifica la espera para invocar la entidad de modelado BPMN que la sucede en el flujo de secuencia. De acuerdo con el esquema mostrado en la Fig. 3.9 (d), la entidad de modelado de BPMN $bpmn\langle\langle S2\rangle\rangle$ debe comenzar T unidades de tiempo después de la ocurrencia del evento ε_{it}. El término de proceso CSP+T que especifica este comportamiento sigue lo descrito en la Fig. 3.13:

$$1.\ \text{ocurrencia de } \varepsilon_{it})\ \frac{\begin{array}{c}P(itime) = (\varepsilon_{it} \bowtie v_{it} \rightarrow SKIP \fatsemi \mathrm{I}(T, v_{it}) \rightarrow SKIP \fatsemi \\ \varepsilon_{S2} \rightarrow SKIP \fatsemi P(S2)) \Box (\varepsilon_{end} \rightarrow SKIP)\end{array}}{\begin{array}{c}v_{it} = s(\varepsilon_{it});\ \ SKIP \fatsemi \mathrm{I}(T, v_{it}) \rightarrow SKIP \fatsemi \\ \varepsilon_{S2} \rightarrow SKIP \fatsemi P(S2) \Box (\varepsilon_{end} \rightarrow SKIP)\end{array}} (\, itime\langle\langle T\rangle\rangle;\ s(\varepsilon_{it}))$$

$$2.\ \mathrm{I}(T, v_{it})\ \text{timeout})\ \frac{\begin{array}{c}\mathrm{I}(T, v_{it}) \rightarrow SKIP \fatsemi \varepsilon_{S2} \rightarrow SKIP \fatsemi \\ P(S2) \Box (\varepsilon_{end} \rightarrow SKIP)\end{array}}{s(\tau);\ \ SKIP \fatsemi \varepsilon_{S2} \rightarrow SKIP \fatsemi P(S2) \Box (\varepsilon_{end} \rightarrow SKIP)} (\, s(\tau) < v_{it} + T\,)$$

$$3.\ \text{ocurrencia de } \varepsilon_{S2})\ \frac{\varepsilon_{S2} \rightarrow SKIP \fatsemi P(S2) \Box (\varepsilon_{end} \rightarrow SKIP)}{s(\varepsilon_{S2});\ \ SKIP \fatsemi P(S2) \Box (\varepsilon_{end} \rightarrow SKIP)} \begin{pmatrix} s(\varepsilon_{S2}) = v_{it} + T; \\ S2 \in ENT \backslash \{start\} \end{pmatrix}$$

Fig. 3.13 SOS del término de proceso correspondiente al evento intermedio temporizado de BPMN.

De acuerdo con la Fig. 3.9 (e), la ejecución de la actividad $bpmn\langle\langle S1\rangle\rangle$ puede ser interrumpida (por la ocurrencia del evento ε_{exc}) T unidades de tiempo después de su comienzo (ocurrencia del evento ε_{S1}), es decir, al finalizar el intervalo de tiempo $[v_{S1}, T]$), si no se pasa a la siguiente actividad (ocurrencia del evento ε_{S2}) antes de agotarse el tiempo T. La operacionalización de la regla de transformación para la especificación de un comportamiento que incluye un *Flujo de excepción temporizado* consiste de dos comportamientos alternativos. El primero, dado por la regla de transformación para una actividad BPMN típica (ver Fig. 3.11), y el segundo, dado por el flujo de excepción especificado en la Fig. 3.14, donde la ocurrencia del evento intermedio temporizado $itime\langle\langle T\rangle\rangle$ interrumpe la ejecución de la actividad $bpmn\langle\langle S1\rangle\rangle$.

Finalmente, para el caso de los *Flujos de mensaje*, representados en la Fig. 3.9 (f), las actividades $bpmn\langle\langle S1\rangle\rangle$ y $bpmn\langle\langle S2\rangle\rangle$ deben intercambiar mensajes dentro del intervalo denotado con la zona sombreada, que representa el lapso de tiempo en el cual ambas actividades están ejecutándose al mismo tiempo; es decir, ambas actividades se están realizando y ni $S1$ ni $S2$ han finalizado en el caso más temprano. La estructura que operacionaliza este intercambio se detalla en la Fig. 3.15, partiendo del supuesto que la actividad $bpmn\langle\langle S1\rangle\rangle$ envía el mensaje $m1$ y después de ser recibido por la actividad $bpmn\langle\langle S2\rangle\rangle$, ésta envía el mensaje $m2$. Sin embargo, podría ser al contrario, lo cual se especifica de la misma forma, sólo que intercambiando $m1$ por $m2$ y $bpmn\langle\langle S1\rangle\rangle$ por $bpmn\langle\langle S2\rangle\rangle$, en donde corresponda,

$$1.\ \text{ocurrencia de } \varepsilon_{S1})\ \frac{\begin{array}{l} P(S1) = (\varepsilon_{S1} \bowtie v_{S1} \rightarrow SKIP \fatsemi \\ \quad \mathrm{I}(S1.ran.max - S1.ran.min, v_{S1} + S1.ran.min).\varepsilon_{S2} \\ \quad \rightarrow (SKIP \bigtriangleup \mathrm{I}(T, v_{S1}) \rightarrow \varepsilon_{exc} \rightarrow SKIP \fatsemi \\ \quad \varepsilon_{S3} \rightarrow SKIP) \fatsemi P(S3)) \Box (\varepsilon_{end} \rightarrow SKIP) \end{array}}{\begin{array}{l} v_{S1} = s(\varepsilon_{S1});\ SKIP \fatsemi \\ \mathrm{I}(S1.ran.max - S1.ran.min, v_{S1} + S1.ran.min).\varepsilon_{S2} \\ \rightarrow (SKIP \bigtriangleup \mathrm{I}(T, v_{S1}) \rightarrow SKIP \fatsemi \varepsilon_{exc} \rightarrow SKIP \fatsemi \\ \varepsilon_{S3} \rightarrow SKIP) \fatsemi P(S3) \Box (\varepsilon_{end} \rightarrow SKIP) \end{array}} \begin{pmatrix} bpmn\langle\langle S1\rangle\rangle; \\ s(\varepsilon_{S1}) \end{pmatrix}$$

$$2.\ \mathrm{I}(T, v_{S1})\ \text{timeout})\ \frac{\begin{array}{l} \mathrm{I}(T, v_{S1}) \rightarrow SKIP \fatsemi \varepsilon_{exc} \rightarrow SKIP \fatsemi \\ \varepsilon_{S3} \rightarrow SKIP) \fatsemi P(S3) \Box (\varepsilon_{end} \rightarrow SKIP) \end{array}}{\begin{array}{l} s(\tau);\ SKIP \fatsemi \varepsilon_{exc} \rightarrow SKIP \fatsemi \\ \varepsilon_{S3} \rightarrow SKIP \fatsemi P(S3) \Box (\varepsilon_{end} \rightarrow SKIP) \end{array}} \begin{pmatrix} itime\langle\langle T\rangle\rangle; \\ s(\tau) < v_{S1} + T \end{pmatrix}$$

$$3.\ \text{ocurrencia de } \varepsilon_{exc})\ \frac{\varepsilon_{exc} \rightarrow SKIP \fatsemi \varepsilon_{S3} \rightarrow SKIP) \fatsemi P(S3) \Box (\varepsilon_{end} \rightarrow SKIP)}{s(\varepsilon_{exc});\ SKIP \fatsemi \varepsilon_{S3} \rightarrow SKIP \fatsemi P(S3) \Box (\varepsilon_{end} \rightarrow SKIP)} (s(\varepsilon_{exc}) = v_{S1} + T)$$

$$4.\ \text{ocurrencia de } \varepsilon_{S3})\ \frac{\varepsilon_{S3} \rightarrow SKIP \fatsemi P(S3) \Box (\varepsilon_{end} \rightarrow SKIP)}{s(\varepsilon_{S3});\ SKIP \fatsemi P(S3) \Box (\varepsilon_{end} \rightarrow SKIP)} \begin{pmatrix} s(\varepsilon_{S2}) > v_{S1} + T; \\ S3 \in ENT \backslash \{start\} \end{pmatrix}$$

Fig. 3.14 SOS del término de proceso correspondiente al evento de excepción temporizado de BPMN.

tal como se muestra en la Fig. 3.16, donde la actividad $bpmn\langle\langle S2\rangle\rangle$ envía el mensaje $m2$ y después de ser recibido por la actividad $bpmn\langle\langle S1\rangle\rangle$, ésta envía el mensaje $m1$. En consecuencia, las dos alternativas para la regla de transformación de los términos de procesos que incluyen la colaboración entre los participantes $Contenedor1$ y $Contenedor2$ quedan estrucutradas de acuerdo a las Figs. 3.15 y 3.16, dependiendo de lo indicado anteriormente[6].

Tal como se muestra en la Fig. 3.15 y en la Fig. 3.16, independientemente del orden de envío de los mensajes, éstos deben ocurrir en la intersección de los intervalos de tiempo $[v_{S1}, v_{S1} + S1.ran.min]$ y $[v_{S2}, v_{S2} + S2.ran.min]$, la cual está representada por la zona sombreada en la Fig. 3.9 (f). En esta zona es seguro que ninguna de las 2 actividades, $bpmn\langle\langle S1\rangle\rangle$ y $bpmn\langle\langle S2\rangle\rangle$, ha podido terminar todavía. Por otro lado, ambas especificaciones muestran un orden para garantizar que el flujo de mensajes se realice correctamente. Primero, las actividades $bpmn\langle\langle S1\rangle\rangle$ y $bpmn\langle\langle S2\rangle\rangle$ son invocadas (ocurrencia de ε_{S1} o ε_{S2}, según corresponda); después, se envía el mensaje que corresponda (ε_{m1} o ε_{m2}, según sea el caso) y se responde con el mensaje correspondiente (ε_{m2} o ε_{m1}, según sea el caso); para finalmente, invocar las actividades $bpmn\langle\langle S3\rangle\rangle$ y $bpmn\langle\langle S4\rangle\rangle$, a través de la ocurrencia de los eventos ε_{S3} o ε_{S4}, en función del caso que corresponda.

Establecida la base formal que sustenta la propuesta que se hace en este trabajo, en el próximo capítulo describimos el punto central de la presente tesis doctoral y que integra algunas de las aportaciones ya descritos en este capítulo.

[6] $MI = \min\{S1.ran.min, S2.ran.min\}$ y $MA = \max\{v_{S1}, v_{S2}\}$.

$$
\begin{array}{l}
P(S1) = (\varepsilon_{S1} \bowtie v_{S1} \to SKIP \fatsemi \mathrm{I}(MI, MA).\varepsilon_{m1}!x \to SKIP \fatsemi \\
\qquad \mathrm{I}(MI, MA).\varepsilon_{m2}?y \to SKIP \fatsemi \\
\qquad \mathrm{I}(S1.ran.max - S1.ran.min, v_{S1} + S1.ran.min).\varepsilon_{S3} \\
\qquad \to SKIP \fatsemi P(S3)) \Box (\varepsilon_{end.1} \to SKIP) \\
P(S2) = (\varepsilon_{S2} \bowtie v_{S2} \to SKIP \fatsemi \mathrm{I}(MI, MA).\varepsilon_{m1}?x \to SKIP \fatsemi \\
\qquad \mathrm{I}(MI, MA).\varepsilon_{m2}!y \to SKIP \fatsemi \\
\qquad \mathrm{I}(S2.ran.max - S2.ran.min, v_{S2} + S2.ran.min).\varepsilon_{S4} \\
\qquad \to SKIP \fatsemi P(S4)) \Box (\varepsilon_{end.2} \to SKIP)
\end{array}
$$

1. ocurrencia de ε_{S1} Y ocurrencia de ε_{S2}

$$
\frac{\begin{array}{l}
v_{S1} = s(\varepsilon_{S1});\ \ SKIP \fatsemi \mathrm{I}(MI, MA).\varepsilon_{m1}!x \to SKIP \fatsemi \\
\mathrm{I}(MI, MA).\varepsilon_{m2}?y \to SKIP \fatsemi \\
\mathrm{I}(S1.ran.max - S1.ran.min, v_{S1} + S1.ran.min).\varepsilon_{S3} \\
\to SKIP \fatsemi P(S3) \Box (\varepsilon_{end.1} \to SKIP) \\
\mathsf{Y} \\
v_{S2} = s(\varepsilon_{S2});\ \ SKIP \fatsemi \mathrm{I}(MI, MA).\varepsilon_{m1}?x \to SKIP \fatsemi \\
\mathrm{I}(MI, MA).\varepsilon_{m2}!y \to SKIP \fatsemi \\
\mathrm{I}(S2.ran.max - S2.ran.min, v_{S2} + S2.ran.min).\varepsilon_{S4} \\
\to SKIP \fatsemi P(S4) \Box (\varepsilon_{end.2} \to SKIP)
\end{array}}{\begin{array}{l}
bpmn\langle\langle S1\rangle\rangle; \\
s(\varepsilon_{S1}) \\
\mathsf{Y} \\
bpmn\langle\langle S2\rangle\rangle; \\
s(\varepsilon_{S2})
\end{array}}
$$

2. ocurrencia de ε_{m1}

$$
\frac{\begin{array}{l}
\mathrm{I}(MI, MA).\varepsilon_{m1}!x \to SKIP \fatsemi \mathrm{I}(MI, MA).\varepsilon_{m2}?y \to SKIP \fatsemi \\
\mathrm{I}(S1.ran.max - S1.ran.min, v_{S1} + S1.ran.min).\varepsilon_{S3} \\
\to SKIP \fatsemi P(S3) \Box (\varepsilon_{end.1} \to SKIP) \\
\mathsf{Y} \\
\mathrm{I}(MI, MA).\varepsilon_{m1}?x \to SKIP \fatsemi \mathrm{I}(MI, MA).\varepsilon_{m2}!y \to SKIP \fatsemi \\
\mathrm{I}(S2.ran.max - S2.ran.min, v_{S2} + S2.ran.min).\varepsilon_{S4} \\
\to SKIP \fatsemi P(S4) \Box (\varepsilon_{end.2} \to SKIP)
\end{array}}{\begin{array}{l}
s(\varepsilon_{m1});\ \ SKIP \fatsemi \mathrm{I}(MI, MA).\varepsilon_{m2}?y \to SKIP \fatsemi \\
\mathrm{I}(S1.ran.max - S1.ran.min, v_{S1} + S1.ran.min).\varepsilon_{S3} \\
\to SKIP \fatsemi P(S3) \Box (\varepsilon_{end.1} \to SKIP) \\
\mathsf{Y} \\
s(\varepsilon_{m1});\ \ SKIP \fatsemi \mathrm{I}(MI, MA).\varepsilon_{m2}!y \to SKIP \fatsemi \\
\mathrm{I}(S2.ran.max - S2.ran.min, v_{S2} + S2.ran.min).\varepsilon_{S4} \\
\to SKIP \fatsemi P(S4) \Box (\varepsilon_{end.2} \to SKIP)
\end{array}} (s(\varepsilon_{m1}) \in [MA, MI])
$$

3. ocurrencia de ε_{m2}

$$
\frac{\begin{array}{l}
\mathrm{I}(MI, MA).\varepsilon_{m2}?y \to SKIP \fatsemi \\
\mathrm{I}(S1.ran.max - S1.ran.min, v_{S1} + S1.ran.min).\varepsilon_{S3} \\
\to SKIP \fatsemi P(S3) \Box (\varepsilon_{end.1} \to SKIP) \\
\mathsf{Y} \\
\mathrm{I}(MI, MA).\varepsilon_{m2}!y \to SKIP \fatsemi \\
\mathrm{I}(S2.ran.max - S2.ran.min, v_{S2} + S2.ran.min).\varepsilon_{S4} \\
\to SKIP \fatsemi P(S4) \Box (\varepsilon_{end.2} \to SKIP)
\end{array}}{\begin{array}{l}
s(\varepsilon_{m2});\ \ SKIP \fatsemi \\
\mathrm{I}(S1.ran.max - S1.ran.min, v_{S1} + S1.ran.min).\varepsilon_{S3} \\
\to SKIP \fatsemi P(S3) \Box (\varepsilon_{end.1} \to SKIP) \\
\mathsf{Y} \\
s(\varepsilon_{m2});\ \ SKIP \fatsemi \\
\mathrm{I}(S2.ran.max - S2.ran.min, v_{S2} + S2.ran.min).\varepsilon_{S4} \\
\to SKIP \fatsemi P(S4) \Box (\varepsilon_{end.2} \to SKIP)
\end{array}} (s(\varepsilon_{m2}) \in [MA, MI])
$$

4. ocurrencia de ε_{S3} Y ocurrencia de ε_{S4}

$$
\frac{\begin{array}{l}
\mathrm{I}(S1.ran.max - S1.ran.min, v_{S1} + S1.ran.min).\varepsilon_{S3} \\
\to SKIP \fatsemi P(S3)) \Box (\varepsilon_{end.1} \to SKIP) \\
\mathsf{Y} \\
\mathrm{I}(S2.ran.max - S2.ran.min, v_{S2} + S2.ran.min).\varepsilon_{S4} \\
\to SKIP \fatsemi P(S4)) \Box (\varepsilon_{end.2} \to SKIP)
\end{array}}{\begin{array}{l}
s(\varepsilon_{S3});\ \ SKIP \fatsemi P(S3) \Box (\varepsilon_{end.1} \to SKIP) \\
\mathsf{Y} \\
s(\varepsilon_{S4});\ \ SKIP \fatsemi P(S4) \Box (\varepsilon_{end.2} \to SKIP)
\end{array}}
\begin{array}{l}
s(\varepsilon_{S3}) \in A; \\
S3 \in ENT \backslash \{start\} \\
\mathsf{Y} \\
s(\varepsilon_{S4}) \in B; \\
S4 \in ENT \backslash \{start\}
\end{array}
$$

Fig. 3.15 SOS del término de proceso correspondiente al flujo de mensaje de BPMN, de la actividad $S1$ a la actividad $S2$.

$$
\begin{array}{l}
P(S2) = (\varepsilon_{S2} \bowtie v_{S2} \to SKIP \fatsemi \mathrm{I}(MI, MA).\varepsilon_{m2}!x \to SKIP \fatsemi \\
\qquad \mathrm{I}(MI, MA).\varepsilon_{m1}?y \to SKIP \fatsemi \\
\qquad \mathrm{I}(S2.ran.max - S2.ran.min, v_{S2} + S2.ran.min).\varepsilon_{S4} \\
\qquad \to SKIP \fatsemi P(S4)) \Box (\varepsilon_{end.2} \to SKIP) \\
P(S1) = (\varepsilon_{S1} \bowtie v_{S1} \to SKIP \fatsemi \mathrm{I}(MI, MA).\varepsilon_{m2}?x \to SKIP \fatsemi \\
\qquad \mathrm{I}(MI, MA).\varepsilon_{m1}!y \to SKIP \fatsemi \\
\qquad \mathrm{I}(S1.ran.max - S1.ran.min, v_{S1} + S1.ran.min).\varepsilon_{S3} \\
\qquad \to SKIP \fatsemi P(S3)) \Box (\varepsilon_{end.1} \to SKIP)
\end{array}
$$

1. ocurrencia de ε_{S2} Y ocurrencia de ε_{S1}

$$
\frac{\begin{array}{l}
v_{S2} = s(\varepsilon_{S2});\ \ SKIP \fatsemi \mathrm{I}(MI, MA).\varepsilon_{m2}!x \to SKIP \fatsemi \\
\mathrm{I}(MI, MA).\varepsilon_{m1}?y \to SKIP \fatsemi \\
\mathrm{I}(S2.ran.max - S2.ran.min, v_{S2} + S2.ran.min).\varepsilon_{S4} \\
\to SKIP \fatsemi P(S4) \Box (\varepsilon_{end.2} \to SKIP) \\
\mathsf{Y} \\
v_{S1} = s(\varepsilon_{S1});\ \ SKIP \fatsemi \mathrm{I}(MI, MA).\varepsilon_{m2}?x \to SKIP \fatsemi \\
\mathrm{I}(MI, MA).\varepsilon_{m1}!y \to SKIP \fatsemi \\
\mathrm{I}(S1.ran.max - S1.ran.min, v_{S1} + S1.ran.min).\varepsilon_{S3} \\
\to SKIP \fatsemi P(S3) \Box (\varepsilon_{end.1} \to SKIP)
\end{array}}{\begin{array}{l}
bpmn\langle\langle S2\rangle\rangle; \\
s(\varepsilon_{S2}) \\
\mathsf{Y} \\
bpmn\langle\langle S1\rangle\rangle; \\
s(\varepsilon_{S1})
\end{array}}
$$

2. ocurrencia de ε_{m2}

$$
\frac{\begin{array}{l}
\mathrm{I}(MI, MA).\varepsilon_{m2}!x \to SKIP \fatsemi \mathrm{I}(MI, MA).\varepsilon_{m1}?y \to SKIP \fatsemi \\
\mathrm{I}(S2.ran.max - S2.ran.min, v_{S2} + S2.ran.min).\varepsilon_{S4} \\
\to SKIP \fatsemi P(S4) \Box (\varepsilon_{end.2} \to SKIP) \\
\mathsf{Y} \\
\mathrm{I}(MI, MA).\varepsilon_{m2}?x \to SKIP \fatsemi \mathrm{I}(MI, MA).\varepsilon_{m1}!y \to SKIP \fatsemi \\
\mathrm{I}(S1.ran.max - S1.ran.min, v_{S1} + S1.ran.min).\varepsilon_{S3} \\
\to SKIP \fatsemi P(S3) \Box (\varepsilon_{end.1} \to SKIP)
\end{array}}{\begin{array}{l}
s(\varepsilon_{m2});\ \ SKIP \fatsemi \mathrm{I}(MI, MA).\varepsilon_{m1}?y \to SKIP \fatsemi \\
\mathrm{I}(S2.ran.max - S2.ran.min, v_{S2} + S2.ran.min).\varepsilon_{S4} \\
\to SKIP \fatsemi P(S4) \Box (\varepsilon_{end.2} \to SKIP) \\
\mathsf{Y} \\
s(\varepsilon_{m2});\ \ SKIP \fatsemi \mathrm{I}(MI, MA).\varepsilon_{m1}!y \to SKIP \fatsemi \\
\mathrm{I}(S1.ran.max - S1.ran.min, v_{S1} + S1.ran.min).\varepsilon_{S3} \\
\to SKIP \fatsemi P(S3) \Box (\varepsilon_{end.1} \to SKIP)
\end{array}} (s(\varepsilon_{m1}) \in [MA, MI])
$$

3. ocurrencia de ε_{m1}

$$
\frac{\begin{array}{l}
\mathrm{I}(MI, MA).\varepsilon_{m1}?y \to SKIP \fatsemi \\
\mathrm{I}(S2.ran.max - S2.ran.min, v_{S2} + S2.ran.min).\varepsilon_{S4} \\
\to SKIP \fatsemi P(S4) \Box (\varepsilon_{end.2} \to SKIP) \\
\mathsf{Y} \\
\mathrm{I}(MI, MA).\varepsilon_{m1}!y \to SKIP \fatsemi \\
\mathrm{I}(S1.ran.max - S1.ran.min, v_{S1} + S1.ran.min).\varepsilon_{S3} \\
\to SKIP \fatsemi P(S3) \Box (\varepsilon_{end.1} \to SKIP)
\end{array}}{\begin{array}{l}
s(\varepsilon_{m1});\ \ SKIP \fatsemi \\
\mathrm{I}(S2.ran.max - S2.ran.min, v_{S2} + S2.ran.min).\varepsilon_{S4} \\
\to SKIP \fatsemi P(S4) \Box (\varepsilon_{end.2} \to SKIP) \\
\mathsf{Y} \\
s(\varepsilon_{m1});\ \ SKIP \fatsemi \\
\mathrm{I}(S1.ran.max - S1.ran.min, v_{S1} + S1.ran.min).\varepsilon_{S3} \\
\to SKIP \fatsemi P(S3) \Box (\varepsilon_{end.1} \to SKIP)
\end{array}} (s(\varepsilon_{m2}) \in [MA, MI])
$$

4. ocurrencia de ε_{S4} Y ocurrencia de ε_{S3}

$$
\frac{\begin{array}{l}
\mathrm{I}(S2.ran.max - S2.ran.min, v_{S2} + S2.ran.min).\varepsilon_{S4} \\
\to SKIP \fatsemi P(S4)) \Box (\varepsilon_{end.2} \to SKIP) \\
\mathsf{Y} \\
\mathrm{I}(S1.ran.max - S1.ran.min, v_{S1} + S1.ran.min).\varepsilon_{S3} \\
\to SKIP \fatsemi P(S3)) \Box (\varepsilon_{end.1} \to SKIP)
\end{array}}{\begin{array}{l}
s(\varepsilon_{S4});\ \ SKIP \fatsemi P(S4) \Box (\varepsilon_{end.2} \to SKIP) \\
\mathsf{Y} \\
s(\varepsilon_{S3});\ \ SKIP \fatsemi P(S3) \Box (\varepsilon_{end.1} \to SKIP)
\end{array}}
\begin{array}{l}
s(\varepsilon_{S4}) \in A; \\
S4 \in ENT \backslash \{start\} \\
\mathsf{Y} \\
s(\varepsilon_{S3}) \in B; \\
S3 \in ENT \backslash \{start\}
\end{array}
$$

Fig. 3.16 SOS del término de proceso correspondiente al flujo de mensaje de BPMN, de la actividad $S2$ a la actividad $S1$.

PROPUESTA

Capítulo 4
Enfoque Formal de Verificación Composicional

Resumen El objeto de este capítulo es presentar el enfoque formal de verificación composicional que proponemos para la verificación de sistemas y de BP. En la sección 4.1 describimos el contexto de nuestro enfoque al dominio del BPM. Luego, en la sección 4.2, presentamos la interpretación del modelado y de la especificación de las propiedades de un sistema o de un BP bajo un dominio semántico común. Seguidamente, en la sección 4.3, introducimos el esquema conceptual de verificación composicional que sustenta nuestra propuesta, seguido de la sección 4.4, donde se muestra la validación de la infraestructura de verificación. Finalmente, en la sección 4.5, describimos el EFVC que resulta de la instanciación del esquema conceptual y que permite la verificación composicional de un sistema y de un BP.

4.1 Contexto de verificación composicional de procesos de negocio con criticidad

En nuestra opinión, la falta de composicionalidad de las técnicas actuales de verificación automática se debe principalmente a problemas semánticos y sintácticos, causados por la integración incorrecta de diferentes formalismos de especificación. Hasta ahora, la integración de notaciones y la integración semántica no ha sido resuelta. Dependiendo de la combinación formal que se haga de *lenguajes de especificación de propiedades* (CTL, ACTL, etc.) y *notaciones de modelado* (UML, BPMN, etc.), la mezcle de notaciones puede inducir a errores semánticos y el resultado de la verificación puede que no refleje fielmente la validez del modelo. Llevar a un dominio semántico común los diferentes formalismos de especificación para que sean interpretados con homogeneidad puede contribuir a solucionar este problema.

Por ello, hemos introducido un esquema conceptual de verificación composicional, sustentado por la transformación de lenguajes de especificación de propiedades y notaciones de modelado que pueden ser interpretados como implementaciones particulares (como son los TBA) de EK. Y hemos propuesto un *Enfoque Formal de Verificación Composicional* (EFVC) automatizable de nuestro esquema conceptual, sobre la base de un conjunto de reglas de transformación (descritas en detalle en la sección 3.4), que uti-

liza herramientas de MC para verificar los componentes individuales de un sistema, y el lenguaje formal composicional basado en CSP, CSP+T, para verificar el comportamiento de todo el sistema a partir de sus componentes verificados, usando un modelo de tiempo discreto (ver sección 3.1). Una versión resumida del esquema para la verificación y del EFVC para Sistemas de Seguridad con Criticidad (*Safety–Critical Systems*, SCS) fue publicada en [143].

Por su parte, dadas las características de los BP (son el resultado de la integración de personas, reglas de negocio, metas de negocio, eventos, información y recursos [67], dejar la validación de éstos como parte del proceso de desarrollo de los BPMS o EIS que los soportan puede resultar una actividad extremadamente costosa y riesgosa para los negocios. Este riesgo se ve incrementado cuando se trata de BP con criticidad, donde el cumplimiento de restricciones temporales es obligatorio para su correcta ejecución. En consecuencia, nuestro propósito es apoyar a los especialistas de BPM con métodos, modelos y herramientas que han probado ser útiles en el dominio de la IS, y hacer posible la verificación automática de BP con criticidad como parte del BPM.

Con el fin de contribuir a la formalización del complejo campo del BPM, consideramos adecuado la incorporación de un modelo ejecutable de los BP, conocido como Modelo de Tareas del BP (*BP Task Model*, BPTM), para lograr este propósito. Un BPTM puede ser considerado como el modelo más básico posible para obtener una descripción completa, exacta, coherente y bien definida del conjunto de actividades y tareas que deben realizar los participantes en un BP para cumplir los objetivos de los usuarios de éste [61, 68]. La formalización de un BPTM establece las bases para la verificación automática de modelos de BP críticos (para más detalles acerca de BP críticos, ver sección 2.6.2).

En nuestra opinión, la falta de verificación de propiedades funcionales y no funcionales de propiedades de los BPTM se debe principalmente a problemas semánticos y sintácticos causada por la incorrecta integración de: (1) la formalización de propiedades (metas y objetivos de negocio, representados por un modelo de BP) en tiempo de diseño [4, 53], y (2) el modelo ejecutable (BPTM) correspondiente en tiempo de ejecución [4, 53] que es necesario derivar de cualquier BP para llevar a cabo la verificación automática de estos modelos.

Proponemos la construcción de un BPTM como un conjunto de términos de proceso de CSP+T, utilizando un conjunto de reglas de transformación de UML a CSP+T [21] o de BPMN a CSP+T [142, 141]. Así, las restricciones temporales de un modelo UML o BPMN en tiempo de diseño pueden ser especificadas por los contructos temporales de CSP+T en tiempo de ejecución. Aspectos comportamentales independientes del tiempo del correspondiente

BPTM pueden ser especificado con el lenguaje de especificación formal no temporizado CSP y, por lo tanto, un BPTM sin restricciones temporales puede ser extraído del BPTM temporizado y verificado con la herramienta FDR2 [71]. Al proceder de esta manera, podemos obtener la verificación completa de un BPTM a partir de un modelo inicial UML o BPMN de una manera confiable. Versiones resumidas del esquema para la verificación y del EFVC para un BPTM modelado con el perfil UML EDOC fueron publicadas en [38, 140], mientras que para un BPTM modelado con BPMN han sido publicadas en [37, 142, 141].

A continuación introducimos las ideas básicas que sustentan el uso de un dominio semántico común para las propiedades y el modelado de un sistema o de un BP.

4.2 Especificando y modelando el comportamiento en un dominio semántico común

CSP+T es el lenguaje de especificación utilizado en nuestro enfoque para verificar si una determinada propiedad es satisfecha por el modelo del sistema o del BP. Consideramos que CSP+T es el lenguaje formal apropiado para describir de forma no ambigua a los componentes de un sistema o los participantes en un BP si queremos capturar su comportamiento fundamental y la secuencia de comunicación en la que éstos están involucrados. Los participantes de un BP se especifican, según nuestro enfoque, mediante un término de proceso de CSP+T, permitiendo la descripción de eventos temporizados complejos a partir de procesos secuenciales simples que son utilizados en la especificación del comportamiento de sistemas concurrentes [92, 184]. La semántica operacional de los cálculos de proceso basados en CSP nos permite interpretar el efecto de ejecutar un constructo sintáctico considerando sólo *un estado a la vez* en la secuencia de ejecución [184, 190] y de esta forma poder calcular las transiciones a las cuales están sujetos los participantes de un BP.

Adicionalmente, el uso de CSP+T añade los modelos de trazas y fallos [190, 184, 92] a la verificación composicional de procesos; dichos modelos son considerados fundamentales para decidir si el modelo que expresa el comportamiento del sistema satisface la especificación de las propiedades no funcionales. Gracias a las definiciones hacia trazas temporizadas y fallos temporizados [190, 224] (ver Definiciones 2.14 y 2.15, respectivamente) podemos verificar el comportamiento temporal especificado en CSP+T considerando a la secuencia de eventos del modelo no temporizado de CSP como la condición necesaria del modelo temporizado especificado con

CSP+T. Estas verificaciones son automatizadas en el FCVA mediante la integración de la herramienta FDR2 [71]. De esta manera, los analistas y diseñadores de BP se benefician de todas las fortalezas de CSP, siendo la efectiva integración de estos avances en el EFVC una aportación para el ámbito de investigación del BPM.

Cabe indicar en este momento que en nuestro enfoque utilizamos indistintamente el término *sistema* para conceptualizar tanto un sistema de software como un sistema conformado por los elementos (o componentes) que definen un BP. Sobre la base de la *Teoría General de los Sistemas* (TGS) [25, 104], ambos pueden ser interpretados bajo los mismos principios y aspectos prácticos, tal como se detalla en el Apéndice D, el cual es una ampliación de la descripción publicada en [143]. Es decir, tanto la definición o arquitectura de un sistema de software, como la definición o arquitectura del sistema que conforman los elementos de un BP, *pueden ser descritos a partir de sus partes (o componentes) y las relaciones que existen entre ellas* [25, 104].

Trabajos como los citados en [67, 101, 120] son una muestra de cómo los conceptos y elementos de modelado de sistemas de software han sido aplicados para conceptualizar y modelar BP. Nosotros, en particular, presentamos en [140, 38] cómo extrapolamos conceptos de sistemas de software de UML para modelar y verificar modelos de BP, como una prueba de que tanto los modelos de BP como los modelos de sistemas pueden ser interpretados de manera similar y utilizar los avances de la IS en el área de BPM.

A continuación se describe cómo logramos especificar bajo el mismo dominio semántico de CSP+T, el modelo de un sistema o de un BP y el conjunto de propiedades que éstos deben exibir, para poder proceder con la verificación composicional que proponemos bajo nuestro enfoque.

4.2.1 Modelo del sistema/proceso de negocio

Para obtener la descripción completa del comportamiento de un sistema, *interpretado* en términos de proceso CSP+T, usamos MEDISTAM–RT [21]. Una serie de vistas del sistema [119], representadas por *diagramas de clases*, *diagramas de estructura compuesta* y *diagramas de máquinas de estado temporizadas* (UML–TSM), se obtienen cuando son aplicados los conceptos de descomposición, abstracción/refinamiento y modelado, en consonancia con la notación de UML–RT. Por lo tanto, el uso de UML–RT nos permite

realizar la *fase de análisis y diseño* de un sistema reactivo[1] y aprovechar los beneficios de la aplicación de *los conceptos de abstración/refinamiento* soportados por UML. A través de la aplicación de las reglas de transformación descrita por MEDISTAM–RT (ver sección 3.4.2), se obtiene el conjunto de términos de proceso CSP+T que corresponden a los UML–TSM que describen el comportamiento de los componentes del sistema.

De acuerdo a nuestro enfoque, los BP críticos son considerados un tipo de sistema 'reactivo', ya que responden a los eventos del entorno de negocios para garantizar la capacidad de una organización para satisfacer a sus clientes. Por ello, tal como se muestra en [38, 140], utlizando una variante de MEDISTAM–RT que parte del perfil UML EDOC (ver sección 3.2.3) logramos obtener los términos de proceso CSP+T que corresponden al BPTM que describen el comportamiento del BP y que son el resultado de la composición de los términos de proceso que especifican el comportamiento de los trabajadores de negocio[2] de un BP. Así, logramos obtener la verificación del BPTM derivado del modelo de un BP usando el perfil UML EDOC a partir de la verificación del comportamiento individual de cada trabajador de negocio.

Para obtener una descripción completa del comportamiento de un modelo BPMN en términos de procesos de CSP+T, aplicamos las reglas de transformación descritas en la sección 3.4.3. Como resultado, cada entidad BPMN (objetos de flujo, objetos de conexión y swimlanes) se hace corresponder a un término sintáctico de proceso secuencial de CSP+T que especifica el comportamiento de los participantes, de acuerdo con una secuencia de eventos y tiempo de ocurrencia discretos. La reglas de transformación permiten dar una semántica temporal de un subconjunto coherente de entidades de modelado de BPMN que se utilizan para la especificación de actividades y eventos del negocio que dependen del tiempo. Mediante una interpretación racional de elementos de modelado de BPMN como términos de procesos de CSP+T se hace viable la verificación composicional del comportamiento global de un BPTM derivado de un modelo BPMN.

4.2.2 Propiedades del sistema/proceso de negocio

Para especificar el conjunto de propiedades que el modelo de un sistema o el modelo de un BP debe exhibir, utilizamos la lógica temporal de intervalos denominada CCTL [188] que nos permite llevar a cabo un razo-

[1] Sistemas que mantienen una interacción constante con su entorno, respondiendo a eventos externos de acuerdo a su estado interno [86, 130].

[2] Los trabajadores de negocio en el perfil UML EDOC equivalen a los participantes en BPMN.

namiento lógico a nivel de intervalos de tiempo, y no en instantes de tiempo (ver sección 2.1.4).

Sin embargo, para poder llevar a cabo la verificación del modelo de un sistema o del modelo de un BP, las propiedades deben expresarse en el mismo dominio semántico de estos modelos. Para ello, se aplica en dos fases, lo siguiente:

1. el algoritmo descrito en la sección 3.5.1, el cual permite construir un TBA discreto semánticamente equivalente a una fórmula CCTL y obtener las estructuras de datos necesarias para representar las fórmulas CCTL que especifican los requisitos del sistema y las restricciones temporales, y
2. las reglas de transformación pautadas por la Definición 3.8, sobre las estructuras de datos obtenidas en la fase anterior, las cuales permiten transformar los TBAs contruidos previamente en términos de procesos en CSP+T.

Como resultado, logramos expresar y razonar acerca del conjunto de propiedades que debe cumplir el modelo del sistema o el BPTM que ha de ser verificado, en el mismo lenguaje de especificación de dichos modelos; es decir, en términos de proceso del lenguaje de especificación CSP+T.

4.3 Esquema conceptual de verificación composicional

Nuestro esquema se basa en el hecho de que el *sistema* C se ha estructurado en varios componentes verificados que trabajan en paralelo, $C = ||_{i:1..n} C_i$, donde cada componente C_i satisface la propiedad ϕ_i, lo que representa la especificación del comportamiento que se espera para el componente.

La Fig. 4.1 muestra el esquema conceptual propuesto para la verificación composicional de un *sistema*; pudiendo ser éste un sistema de software o un BP. La primera versión que presentamos de este esquema estuvo enfocado al modelado y a la verificación de Sistemas con Criticidad en Seguridad (*Safety–Critical Systems*, SCS) [136, 143]; sin embargo, como hemos mostrado en publicaciones posteriores (ver [140, 38, 37, 142] para más detalles), el mismo puede ser aplicado en el ámbito de BPM.

Cada componente debe satisfacer la *invariante* (ψ_i), la cual representa el comportamiento de los otros componentes del sistema con respecto a C_i. Se usa el símbolo especial $\neg\delta$ para denotar que el bloqueo (*deadlock*) del componente (es decir, 'caer' en un estado sin ninguna transición de salida) no puede ser alcanzado. La propiedad ϕ y la invariante ψ que son satisfechas por el sistema C se obtienen a partir de las propiedades locales ϕ_i

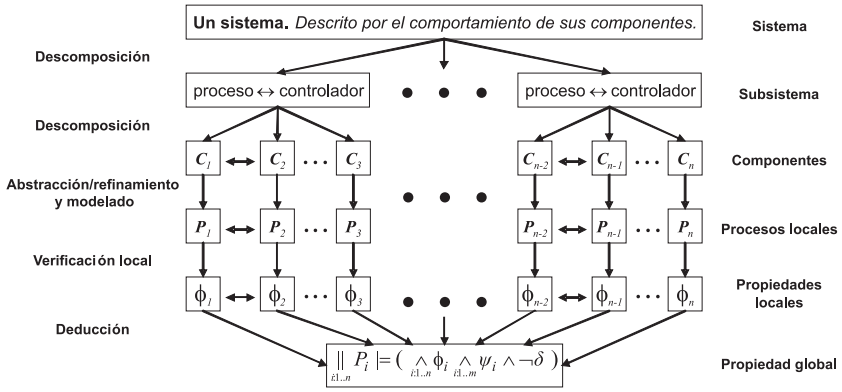


Fig. 4.1 Esquema conceptual para la verificación composicional.

($\bigwedge_{i:1..n} \phi_i \Rightarrow \phi$) y de los invariantes locales ψ_i ($\bigwedge_{i:1..n} \psi_i \Rightarrow \psi$), respectivamente.

De esta manera podemos aplicar nuestro esquema composicional a propiedades críticas de seguridad (obtenidas como combinaciones de los operadores temporales AG de forma natural), propiedades de vivacidad, denotadas como $\mathsf{AG}(req \rightarrow \mathsf{AF}sat)$, expresando que la satisfacción de *sat* es inevitable dado *req*; es decir, *req* es condición suficiente para que ocurra *sat* en el futuro; propiedades de libertad de bloqueo (generalmente escritas como $\mathsf{AGEX}true$, y denotadas en nuestro enfoque con el símbolo especial $\neg\delta$), y propiedades de justicia (escritas en CCTL como $\mathsf{AGAF}\phi$). En contraposición, fórmulas de lógica temporal que expresan la posible ocurrencia de ϕ en el futuro (escritas en CCTL como $\mathsf{EF}\phi$ o $\mathsf{EG}\phi$); es decir, propiedades de alcanzabilidad expresadas por la combinación de los operadores EF o EG, no son preservadas por la composicionalidad [174] (ver sección 2.5.2).

Como resultado, podemos obtener la verificación completa de un sistema (de software o un BP) mediante el Teorema 4.1 (definido más adelante), de acuerdo a la relación (4.1). La aplicación práctica de la relación (4.1) incluye (manualmente) realizar un proceso de 'chequeo de satisfacción' inductivo en el rango del número de componentes ($i : 1, \ldots, n$) de un sistema. Esta prueba se encuentra automatizada en nuestra propuesta a través del uso del comprobador de modelos FDR2 [71].

Antes de presentar el Teorema 4.1 y su validación (ver sección 4.4, más adelante), queremos resaltar los aspectos que éste abarca. El Teorema 4.1 se basa en el orden temporal de la comunicación (de conformidad con los requisitos de sincronización preestablecidos) que debe ser preservado por el comportamiento del sistema (o BP). Para lograr esto, se incluye como parte de su validación (o prueba) las siguientes condiciones, las cuales co-

rresponden a una ejecución natural de SCS o BP críticos correctos y bien coordinados: (1) las situaciones de deadlock no puede surgir si el orden de ejecución de las tareas de los componentes del sistema o de los participantes del BP se preservan y se respetan las restriciones temporales en las comunicaciones; (2) la composicionalidad de los invariantes (es decir, el comportamiento del entorno de los componentes del sistema o de los participantes del BP) está garantizada siempre y cuando se mantengan locales a cada componente o participante y los alfabetos de comunicación (es decir, el conjunto de mensajes intercambiados) sean disjuntos $\bigcap_{i:1..n} \boldsymbol{\Sigma}_i = \varnothing$ and $\bigcap_{i:1..n} \boldsymbol{\Omega}_i = \varnothing$); (3) las propiedades locales de los componentes o participantes se conservan por la composición paralela del uso autónomo/independiente de las variables locales o los nombres por cada componente o participante ($\bigcap_{i:1..n} \mathbf{L}(TBA(C_i)) = \varnothing$); y, finalmente, (4) las propiedades de vivacidad son "naturalmente" composicionales y las propiedades de seguridad combinadas con los invariantes locales también son composicionales.

Por otro lado, tómese en cuenta que realizamos la demostración del Teorema 4.1 de acuerdo con la semántica del TBA, con el fin de abstraernos de los detalles de implementación que cada uno de los cálculos de proceso (tal como la utilizada por el FCVA descrito próximamente en la sección 4.5) hace para el operador de concurrencia ($\|$). Para describir el comportamiento de los componentes del sistema o de los participantes del BP, usamos el TBA debido a que es el más adecuado para nuestros propósitos, tal como indicamos en la sección 3.2.3. El TBA nos da una sólida base matemática para el razonamiento acerca de las especificaciones, los diseños e implementaciones, y puede ser utilizado tanto para la investigación teórica como para las aplicaciones prácticas [92, 11], tal como se muestra en esta tesis doctoral. La capacidad del TBA como representación abstracta gráfica de distintas implementaciones es uno de los beneficios de los formalismos basados en autómatas. En las Fig. 3.1 y 3.2 (capítulo 3) se muestran las representaciones gráficas de una fórmula CCTL y un término de proceso CSP+T, respectivamente, como ejemplo de la capacidad de abstracción que tiene el TBA para representar la implementación de los formalismos utilizados por nuestro enfoque. A continuación, el enunciado del Teorema 4.1[3] y su demostración. Recuérdese que nosotros utilizamos indistintamente el término sistema para conceptualizar tanto un sistema de software como un sistema conformado por los elementos (o componentes) que definen un BP.

[3] Escribimos $TBA(C) \vDash \phi$ cuando el $TBA(C)$ mantiene la propiedad ϕ para todos sus estados iniciales, es decir, $\forall\, s \in I \Rightarrow (TBA(C), s) \vDash \phi$. Para los invariantes tenemos $TBA(C) \vDash \psi$ cuando para todos los $s \in S$ se mantiene $(TBA(C), s) \vDash \psi$. Para denotar que $TBA(C)$ no contiene algún bloqueo, escribimos $TBA(C) \vDash \neg\delta$.

Teorema 4.1. ***Verificación composicional de sistemas.*** *Sea el sistema C estructurado en varios componentes trabajando en paralelo, $C = \|_{i:1..n} C_i$. Para un conjunto de $TBA(C_i)$ que describen el comportamiento de los componentes C_i, las propiedades ϕ_i, los invariantes ψ_i, y el bloqueo δ, con $\bigcap_{i:1..n} \boldsymbol{\Sigma}_i = \varnothing$, $\bigcap_{i:1..n} \boldsymbol{\Omega}_i = \varnothing$, y $\bigcap_{i:1..n} \mathbf{L}(TBA(C_i)) = \varnothing$, la siguiente condición se comple:*

$$TBA(C) \vDash (\phi \wedge \psi \wedge \neg\delta) \Leftrightarrow \Big|\Big|_{i:1..n} TBA(C_i) \vDash \bigwedge_{i:1..n} (\phi_i \wedge \psi_i) \wedge \neg\delta, \quad (4.1)$$

donde $TBA(C) = \|_{i:1..n}\ TBA(C_i)$.

Demostración. Asumimos que el comportamiento del componente C_i puede ser descrito por el $TBA(C_i)$. Además, podemos asociar un término de proceso $P(C_i)$ de algún cálculo de proceso (CSP, CCS, ACP, etc.) con un componente C_i, a fin de:

$$P(C_i) \models (\phi_i \wedge \psi_i) \wedge \neg\delta\ .$$

Por el operador de composición paralela del cálculo de proceso, la siguiente relación debe ser satisfecha:

$$P(C) = \Big|\Big|_{i:1..n} P(C_i)\ .$$

Por la relación de simulación (ver Definición 3.5),

$$P(C) \preceq TBA(C) \Rightarrow (P(C) \models \phi \wedge \psi \wedge \neg\delta \Rightarrow TBA(C) \models \phi \wedge \psi \wedge \neg\delta), \text{ y}$$

$$P(C_i) \preceq TBA(C_i) \Rightarrow (P(C_i) \models (\phi_i \wedge \psi_i) \wedge \neg\delta \Rightarrow TBA(C_i) \models (\phi_i \wedge \psi_i) \wedge \neg\delta)\ .$$

Si los $TBA(C_i)$ satisfacen,

1. $\bigcap_{i:1..n} \boldsymbol{\Sigma}_i = \varnothing$ y $\bigcap_{i:1..n} \boldsymbol{\Omega}_i = \varnothing$, y
2. $\bigcap_{i:1..n} \mathbf{L}(TBA(C_i)) = \varnothing$,

podemos afirmar que éstos pueden ser compuestos y, por el Lema 4.1 (ver sección 4.4), escribimos:

$$\Big|\Big|_{i:1..n} TBA(C_i) \models \bigwedge_{i:1..n} (\phi_i \wedge \psi_i) \wedge \neg\delta$$

Dado que la relación de satisfacción es cerrada con respecto al operador de conjunción, concluimos $\bigwedge_{i:1..n} \phi_i \Rightarrow \phi$, $\bigwedge_{i:1..n} \psi_i \Rightarrow \psi$, y por consiguiente:

$$P(C) \models \phi \wedge \psi \wedge \neg\delta \Leftrightarrow \Big|\Big|_{i:1..n} P(C_i) \models \bigwedge_{i:1..n} (\phi_i \wedge \psi_i) \wedge \neg\delta\ .$$

La condición suficiente $\phi \Rightarrow \bigwedge_{i:1..n} \phi_i$, $\psi \Rightarrow \bigwedge_{i:1..n} \psi_i$, puede ser demostrada considerando que aun en el caso donde todos los componentes C_i presenten un

comportamiento diferente, ϕ y ψ podrán ser definidas como la conjunción de las propiedades locales de cada componente (ϕ_i, ψ_i). □

4.4 Validación de la infraestructura de verificación composicional

En esta sección procedemos a detallar la prueba matemática de la base teórica de nuestro enfoque, la cual está orientada a demostrar el Teorema 4.1 que nos permite obtener una verificación completa del sistema o del BP a partir de sus componentes o participantes, respectivamente, verificados individualmente. Para obtener la formalización que se muestra en esta sección, nos centraremos en el modelo de comportamiento y la estructura de sincronización [64] de los componentes del sistema o participantes del BP para excluir los errores de diseño al nivel de abstracción en el cual se desarrolla la demostración (ver Apéndice D para mayores detalles).

4.4.1 Definiciones básicas

Los TBA a los cuales hacemos referencia en esta sección corresponden a los descritos a través de la Definición 3.4.

Por razones prácticas, usamos Σ_i, S_i, I_i, C_i, R_i, y F_i para denotar los elementos correspondientes de $\mathbf{A}_i$. Dos TBA $\mathbf{A}$ y $\mathbf{A}'$ *pueden ser compuestos sin sincronizaciones no deseadas* si tienen distintos conjuntos de entrada y salida ($\Sigma \cap \Sigma' = \varnothing$ y $\Omega \cap \Omega' = \varnothing$). Si también se tiene $\Sigma \cap \Omega' = \varnothing$ y $\Sigma' \cap \Omega = \varnothing$, se dice que son *ortogonales* entre si. Cualquier TBA $\mathbf{A}$ con $\Sigma = \Omega$ es llamado *cerrado*.

Los componentes interactúan con el entorno del sistema a través de sus interfaces. En consecuencia, la composición del comportamiento de las interfaces describen tanto el comportamiento completo del componente como el comportamiento general del sistema. Nuestro EFVC supone que los requisitos del sistema pueden ser obtenidos por medio de una serie de componentes que se comunican a través de sus interfaces.

4.4.2 Descomposición/composición del sistema

Tomando en cuenta que el sistema C ha sido dividido en varios componentes trabajando en paralelo, $C_1, \ldots, C_i$, representados por los TBAs $\mathbf{A}_1^C, \ldots, \mathbf{A}_i^C$,

se ha de definir la composición (paralela) de TBAs como el resultado de una ejecución síncrona [49] de los TBAs que representan todos los componentes del sistema ejecutándose en paralelo. Para dar una interpretación de la composición de componentes en la semantica de los TBA (los cuales son una representación de las EK), exponemos a continuación la Definición 4.1.

Definición 4.1. Composición de TBA. Para dos TBA $\mathbf{A} = \langle \Sigma, \Omega, S, I, C, R, F \rangle$, y $\mathbf{A}' = \langle \Sigma', \Omega', S', I', C', R', F' \rangle$, que pueden ser compuestos el uno con el otro ($\Sigma \cap \Sigma' = \varnothing$ y $\Omega \cap \Omega' = \varnothing$), definimos su composición paralela denotada por $\mathbf{A} \| \mathbf{A}'$ como el TBA $\mathbf{A}'' = \langle \Sigma'', \Omega'', S'', I'', C'', R'', F'' \rangle$ con $S'' = S \times S'$, $\Sigma'' = \Sigma \cup \Sigma'$, $\Omega'' = \Omega \cup \Omega'$, $I'' = I \times I'$, $C'' = C \cup C'$, $F'' = F \times F'$ y $((s_1, s_1'), E'', O'', (s_2, s_2')) \in R''$ si y sólo si existen $(s_1, E, O, s_2) \in R$ y $(s_1', E', O', s_2') \in R'$ con $E'' = E \cup E'$ y $O'' = O \cup O'$, donde $E \subseteq \Sigma$, $O \subseteq \Omega$, $E' \subseteq \Sigma'$, $O' \subseteq \Omega'$, $E'' \subseteq \Sigma''$ y $O'' \subseteq \Omega''$. Adicionalmente, $(E \cap \Omega') = O'$ y $(E' \cap \Omega) = O$ debe mantenerse. S'' y R'' se han ajustado para excluir todas las combinaciones de estados y transiciones no alcanzables.

Una transición en R'' es una combinación de dos transiciones en cada autómata si y sólo si todas las entradas requeridas por el otro lado son correspondidas ($(E \cap \Omega') = O'$ y $(E' \cap \Omega) = O$); es decir, conforman el alfabeto de comunicación, y el etiquetado compartido (definido en la siguiente sección) de los estados objetivo es idéntico. Las señales de entrada externa y de salida son simplemente la unión de ambos TBA. Si queremos hacer abstracción de señales de entrada interna y de salida sólo tenemos que utilizar la restricción para Σ_1 / Ω_1 con $\Sigma_1 = \Sigma'' - (\Omega \cup \Omega')$ y $\Omega_1 = \Omega'' - (\Sigma \cup \Sigma')$.

4.4.3 Refinamiento

Se debe garantizar la correctitud de la fase de diseño para permitir la preservación del comportamiento abstracto establecido por las propiedades hasta el comportamiento detallado ofrecido por los términos del proceso que modelan los componentes de software del sistema. Ver nuestra definición de refinamiento dentro del contexto de los aspectos prácticos del EFVC en la sección D.3 del Apéndice D. Así, la Definición 4.2 de refinamiento (Ver nuestro concepto de refinamiento dentro del contexto de los aspectos prácticos del EFVC en la sección D.3 del Apéndice D) es lo suficientemente fuerte para preservar la ausencia de bloqueos y para soportar resultados de refinamientos posteriores como parte del proceso de diseño del sistema.

Definición 4.2. Refinamiento. Para dos TBAs $\mathbf{A}$ y $\mathbf{A}'$ llamamos a $\mathbf{A}$ un refinamiento de $\mathbf{A}'$ denotado por $\mathbf{A}' \sqsubseteq \mathbf{A}$ si y sólo si existe la relación $\Psi \subseteq S \times S'$ con $\forall\, q \in I\ \exists\, q' \in I' : (q, q') \in \Psi$ y para todo $(s_1, s_1') \in \Psi$ se mantiene:

$$\forall(s_1, E, O, s_2) \in R \;\; \exists(s_1', E, O, s_2') \in R' \;:\; (s_2, s_2') \in \Psi, \tag{4.2}$$

$$\forall(s_1', E', O', s_3') \in R' \;\; \exists(s_1, E', O', s_3) \in R. \tag{4.3}$$

En general, las condiciones para un refinamiento en el EFVC asume que el refinamiento del comportamiento preserva una similaridad de composición estructural [64, 136]. Para mantener una composición de estructura independiente de la noción de refinamiento de comportamiento deben emplearse los conceptos más abstractos de trazas y fallos [92] en lugar de comparar las estructuras de transición en si mismas derivadas de un refinamiento de la composición estructural. Esto restringe la aplicación general para dos TBA arbitrarios. En el EFVC obligamos a que una relación de refinamiento comportamental deba establecerse también la similaridad de composición estructural que naturalmente es establecida a partir de la fase de análisis y diseño; es decir, la composición estructural del sistema se conserva durante todo la fase de análisis y diseño (véase [136] para más detalles).

4.4.4 Propiedades y verificación local

Para llevar a cabo la verificación, obtenemos previamente los TBAs que representan las propiedades que deben verificarse; es decir, los estados de los TBAs contienen las AP usadas para describir propiedades básicas. Usando la función de etiquetado $L_i : S \rightarrow \wp(p_i)$, los TBAs $\mathbf{A}_i$ y cualquiera de sus estados $s \in S_i$ son etiquetados con todas las proposiciones $p_i \subseteq AP$ que se cumplen en dichos estados, tal como pauta la Definición 4.3.

Definición 4.3. Etiquetado. Un TBA $\mathbf{A}_i = \langle \Sigma_i, \Omega_i, S_i, I_i, C_i, R_i, F_i \rangle$ es en consecuencia extendido a $\mathbf{A}_i = \langle \Sigma_i, \Omega_i, S_i, I_i, C_i, R_i, L_i, F_i \rangle$ por la función de etiquetado $L_i : S \rightarrow \wp(p_i)$, donde el conjunto de etiquetas $\mathbf{L}(\mathbf{A}_i)$ denota el conjunto de todas las proposiciones etiquetadas p_i, y $\mathbf{L}(\phi)$ denota los subconjuntos del conjunto de las proposiciones básicas AP que son empleadas en las fórmulas CCTL.
Para un determinado etiquetado requerimos que $AP = AP'$ y que L y L' sean preservados por Ψ: $(s, s') \in \Psi \Rightarrow L(s) = L'(s')$.

La realización del comportamiento de cada uno de los componentes puede ser verificada usando una herramienta de MC común.

4.4.5 Verificación composicional

Debido a que el modelo del sistema (es decir, el conjunto de términos de proceso derivados de los componentes que deben verificarse) representa un comportamiento más detallado del comportamiento abstracto especificado por una propiedad, como resultado de una fase de análisis y diseño sistemática [21], nos enfocamos en las propiedades y los invariantes que son preservadas a lo largo del refinamiento y la composición con *etiquetados disjuntos*, lo cual se establece en la Definición 4.4. Nótese que en el EFVC los conjuntos de etiquetado disjuntos de múltiples autómatas se usan para emular los efectos de nombres locales; es decir, la autonomía/independencia de variables locales o nombres para cada componente C_i, y por ende, para cada TBA $\mathbf{A}_i$. Recuérdese que nuestro objetivo principal es hacer posible la verificación del comportamiento del sistema completo a partir de sus componentes verificados.

Definición 4.4. Composicionalidad de propiedades. Una propiedad ϕ es composicional si y sólo si para cualesquiera dos TBA $\mathbf{A}_1$, $\mathbf{A}'_1$, y $\mathbf{A}_2$ con $\mathbf{L}(\mathbf{A}_2) \cap \mathbf{L}(\phi) = \varnothing$ se mantiene

$$(\mathbf{A}_1 \vDash \phi) \Rightarrow ((\mathbf{A}_1 \| \mathbf{A}_2 \vDash \phi) \vee \mathbf{A}_1 \| \mathbf{A}_2 \vDash \delta)) \text{ y} \tag{4.4}$$

$$((\mathbf{A}_1 \sqsubseteq \mathbf{A}'_1) \wedge (\mathbf{A}'_1 \vDash \phi)) \Rightarrow (\mathbf{A}_1 \vDash \phi) \tag{4.5}$$

Propiedades locales son preservadas por la composición paralela cuando el etiquetado es disjunto, como indica el Lema 4.1.

Lema 4.1. *Para dos TBAs* $\mathbf{A}_1$ *y* $\mathbf{A}_2$ *y propiedades* ϕ_1 *y* ϕ_2 *con* $\Sigma_1 \cap \Omega_2 = \varnothing$, $\Sigma_2 \cap \Omega_1 = \varnothing$, $\mathbf{L}(\mathbf{A}_1) \cap \mathbf{L}(\mathbf{A}_2) = \varnothing$ *se mantiene:*

$$((\mathbf{A}_1 \vDash \phi_1) \wedge (\mathbf{A}_2 \vDash \phi_2)) \Rightarrow (A_1 \| A_2 \vDash \phi_1 \wedge \phi_2). \tag{4.6}$$

Demostración. Como $\mathbf{L}(\mathbf{A}_1) \cap \mathbf{L}(\mathbf{A}_2) = \varnothing$, podemos concluir que si ϕ_2 se mantiene, ésta no es influenciada por $\mathbf{A}_1$. Dada la asunción $\Sigma_1 \cap \Omega_2 = \varnothing$, $\Sigma_2 \cap \Omega_1 = \varnothing$, $\mathbf{L}(\mathbf{A}_1) \cap \mathbf{L}(\mathbf{A}_2) = \varnothing$, ambos TBA son ejecutados independientemente en paralelo y así $\mathbf{A}_2 \vDash \phi_2$ implicará $A_1 \| A_2 \vDash \phi_2$, y $\mathbf{A}_1 \vDash \phi_1$ implicará $A_1 \| A_2 \vDash \phi_1$. Por consiguiente, $A_1 \| A_2 \vDash \phi_1 \wedge \phi_2$ ha sido probado. □

Por otra parte, también es necesario que la composición paralela conserve el refinamiento. Este hecho nos permitirá sustituir más tarde autómatas refinados (derivados de los términos de proceso refinados) por unos más abstractos que preserven las propiedades especificadas, tal como establece el Lema 4.2.

Lema 4.2. *Para dos TBAs* $\mathbf{A}_1$ *y* $\mathbf{A}_2$ *que pueden ser compuestos, y cualquier TBA* $\mathbf{A}'_2$ *se mantiene*

$$\mathbf{A}_2 \sqsubseteq \mathbf{A}'_2 \Rightarrow (\mathbf{A}_1 \| \mathbf{A}_2 \sqsubseteq \mathbf{A}_1 \| \mathbf{A}'_2). \tag{4.7}$$

Demostración. Para $\mathbf{A} = \mathbf{A}_1 \| \mathbf{A}_2$ y $\mathbf{A}' = \mathbf{A}_1 \| \mathbf{A}'_2$ podemos, a partir de la relación Ψ conseguida por el refinamiento $\mathbf{A}_2 \sqsubseteq \mathbf{A}'_2$, deriva una relación Ψ' requerida para el refinamiento $\mathbf{A} \sqsubseteq \mathbf{A}'$ $(\mathbf{A}_1 \| \mathbf{A}_2 \sqsubseteq \mathbf{A}_1 \| \mathbf{A}'_2)$ como sigue: Para todo $(s_1, s'_2) \in S_1 \times S'_2$ y $(s_2, s'_2) \in \Psi$ agrega $((s_1, s_2), (s_1, s'_2))$ a Ψ'. Dada la composición de R respecto a R' a partir de R_1 y R_2 respecto a R'_2 podemos fácilmente probar la relación 4.7. □

También se requiere que adicionalmente el refinamiento preserve la ausencia de bloqueo y los invariantes, según el Lema 4.3:

Lema 4.3. *Para dos TBAs* $\mathbf{A}$ *y* $\mathbf{A}'$ *que pueden ser compuestos, con* $\mathbf{A} \sqsubseteq \mathbf{A}'$*, se mantiene*

$$\mathbf{A}' \vDash \neg\delta \Rightarrow \mathbf{A} \vDash \neg\delta \text{ y} \tag{4.8}$$

$$\mathbf{A}' \vDash \psi^{C'} \Rightarrow \mathbf{A} \vDash \psi^{C} \tag{4.9}$$

Demostración. Si no se mantiene $\mathbf{A}' \vDash \neg\delta \Rightarrow \mathbf{A} \vDash \neg\delta$, podemos concluir que el refinamiento $\mathbf{A}$ el cual contiene un estado de bloqueo $s \in S$ y un comportamiento original $\mathbf{A}'$ el cual no contiene algún estado de bloqueo. Dada la relación 4.2 podemos identificar cualquier estado alcanzable s y así también al menos un estado $s' \in S'$ con $(s, s') \in \Psi$ (la relación Ψ conseguida por el refinamiento asumido). Como $\mathbf{A}'$ no contiene un estado de bloqueo, s' tiene al menos una transición de salida y según la relación 4.8 también contiene s. Esto contradice nuestra asunción inicial que s es un estado de bloqueo, provando nuestra aseveración por contradicción.
El mismo argumento se mantiene para los invariantes y así la condición 4.9 también queda probada. □

Por último, tenemos que probar que las propiedades composicionales, los invariantes y la ausencia de bloqueo, son preservadas, de acuerdo al Lema 4.4.

Lema 4.4. *Para dos TBAs* $\mathbf{A}_1$ *y* $\mathbf{A}_2$ *que pueden ser compuestos, y cualquier autómata* $\mathbf{A}'_2$ *con* $A_2 \sqsubseteq A'_2$*,* $\Sigma_1 \cap (\Omega_2 - \Omega'_2) = \varnothing$*,* $\Omega_1 \cap (\Sigma_2 - \Sigma'_2) = \varnothing$*, y* $\mathbf{L}(\mathbf{A}_1) \cap (\mathbf{L}(\mathbf{A}_2) - \mathbf{L}(\mathbf{A}'_2)) = \varnothing$ *y cualquier propiedad composicional* ϕ*, se mantiene*

$$(\mathbf{A}_1 \| \mathbf{A}'_2 \vDash \phi \wedge \psi \wedge \neg\delta) \Rightarrow (\mathbf{A}_1 \| \mathbf{A}_2 \vDash \phi \wedge \psi \wedge \neg\delta) \tag{4.10}$$

Demostración. A partir de $\Sigma_1 \cap (\Omega_2 - \Omega'_2) = \varnothing$, $\Omega_1 \cap (\Sigma_2 - \Sigma'_2) = \varnothing$ se sigue que $\mathbf{A}_2$ agrega a $\mathbf{A}'_2$ sólo las señales de E/S que no interfieren con $\mathbf{A}_1$ y así $\mathbf{A}_1 \| \mathbf{A}_2$ contiene el mismo conjunto de estados alcanzables y transiciones, y así $\mathbf{A}_1 \| \mathbf{A}_2 \vDash \psi \wedge \neg\delta$. Como ϕ es interpretada sólo sobre estados y el etiquetado es idéntico para $\mathbf{L}(\phi) \subseteq \mathbf{L}(\mathbf{A}'_2)$, ϕ debe también mantenerse. Así la relation 4.10 es probada. □

4.4.6 Teorema de la verificación composicional

Partiendo de las definiciones y lemas descritos anteriormente, es posible demostrar el Teorema 4.1, tal como se presentó en la sección 4.3. De acuerdo al Teorema 4.1, el EFVC nos permite verificar que el comportamiento es un sistema es *correcto*[4] si el comportamiento de cada uno de sus componentes es correcto sin construir el espacio de estados resultante de la composición paralela de los componentes del sistema objetivo. Sólo es necesario verificar la correctitud sintáctica (estructura de composición) de todo el sistema y la correctitud del comportamiento de los componentes del sistema.

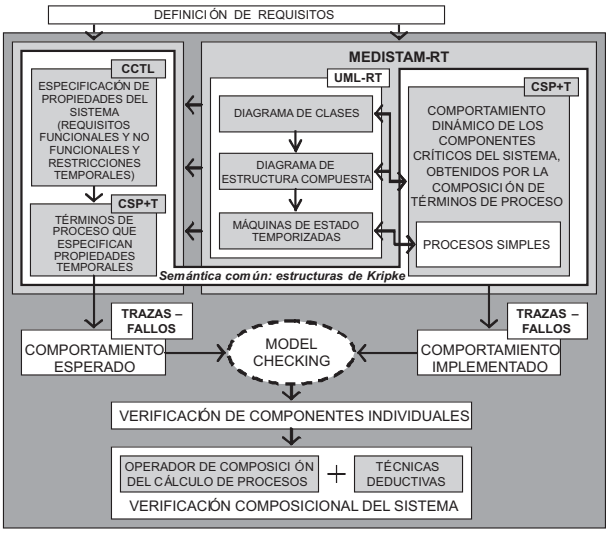
4.4.7 Comentario final

En concordancia con lo que ya comentamos en la sección 2.5.1, de acuerdo con [78, 174] la ausencia de deadlocks, los invariantes, los límites de tiempo superior e inferior y las fórmulas ACTL (el subconjunto de fórmulas CCTL que sólo utilizan el cuantificador de camino A y tienen la negación ($\neg$) sólo en las proposiciones —forma normal), han demostrado ser composicionales. En contraste con este resultado, las fórmulas de lógica temporal cuyo estado explícitamente establece que un estado específico puede ser eventualmente rechazado (abstrayéndose de los posibles efectos del no–determinismo), no pueden ser preservadas [78, 174] si se aplica la composicionalidad. Es más, si $\mathsf{A}\phi$ no se verifica, entonces existe una ejecución que no satisface ϕ, y por lo tanto se satisface $\neg\phi$; es decir, $\mathsf{A}\phi$ y $\neg\mathsf{E}\neg\phi$ son equivalentes [22].

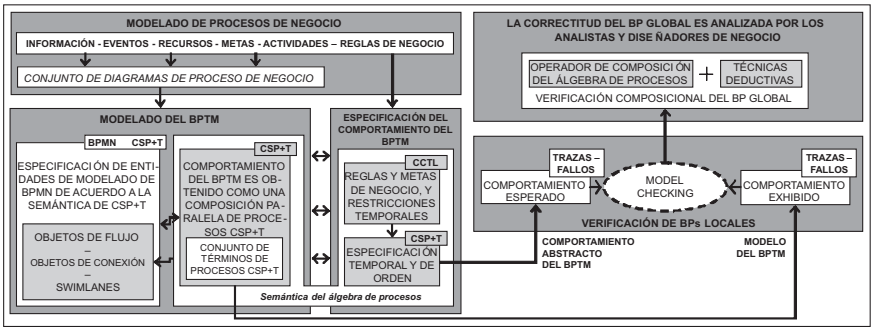
4.5 Descripción del Enfoque Formal de Verificación Composicional

Basados en los conceptos e ideas anteriores, se propone una instanciación del esquema conceptual, conocido como EFVC, que integra los formalismos descritos en la sección 4.2 y el esquema presentado en la sección 4.3. Las versiones gráficas del EFVC que muestran nuestra visión de integración son presentadas en la Fig. 4.2, las cuales serán discutidas en la sección 4.5.1. Porteriormente, en la sección 4.5.2, describiremos el flujo de las actividades para llevar a cabo la verificación del comportamiento de un sistema o de un BP.

[4] También llamado *semánticamente correcto* [66].



(a) EFVC para la verificación de SCS.



(b) EFVC para la verificación de BPTM.

Fig. 4.2 Nuestra visión integrada de verificación composicional.

4.5.1 Integración de formalismos para verificar

La fundamentación del EFVC es que la corrección del comportamiento de los componentes de un SCS (ver Fig. 4.2 (a)) y/o del comportamiento de los participantes en un BPTM (ver Fig. 4.2 (b)) puede ser verificado de forma individual, por separado, basado en el comportamiento de las comunicaciones especificado por sus interfaces. Posteriormente, utilizando los resultados de la verificación de los distintos componentes o participantes locales, llevar a cabo la verificación del comportamiento del sistema o del BP, respectivamente. Nuestra instanciación utiliza el lenguaje de especificación CSP+T, que tiene una forma simple pero poderosa de composición dada

por los operadores de composición concurrente y de ocultamiento. De esta forma aprovechamos de manera completa las fortalezas que los cálculos de proceso basados en CSP aportan a la formalización del comportamiento y de los aspectos temporales de un SCS o de un BPTM, tanto en tiempo de diseño como en tiempo de ejecución. De esta manera podemos integrar herramientas de MC a los procesos de desarrollo de SCS y de BPM.

Como la definición de los requisitos del sistema o de las reglas, metas, objetivos, etc., del BP, se considera fuera del ámbito del EFVC (parte superior de la Fig. 4.2 (a)) y parte superior izquierda de la Fig. 4.2 (b), respectivamente), nuestra recomendación es que se incluyan especificaciones formales (tanto del modelo del sistema o del BPTM, como de sus propiedades) desde el comienzo del desarrollo de SCS o del diseño del BP para reducir el trabajo de transformar dichas especificaciones a un lenguaje formal que permita la verificación. Tanto la descripción formal del comportamiento del sistema o del BP como la especificación de sus propiedades, deben estar orientadas por los requisitos (planteados por los stakeholders) del sistema o por los analistas y diseñadores de BP.

Los formalismos utilizados para *Modelar el sistema o BPTM* (véase parte media–derecha de la Fig. 4.2 (a) y parte inferior–izquierda de la Fig. 4.2 (b), respectivamente) se basan en el hecho de que una descripción completa del comportamiento del sistema o del BPTM no se puede obtener sólo con la estructura del sistema o del BP sin tener en cuenta el comportamiento dinámico y las restricciones temporales representados por los componentes o participantes y las restricciones temporales referidas a la colaboración (es decir, al intercambio de mensajes) entre componentes del sistema o participantes del BPTM.

La descripción completa del comportamiento temporal del sistema o del BPTM se obtiene mediante la aplicación de las reglas de transformación pautadas por MEDISTAM–RT o por nuestra propuesta semántica de tiempo para algunos elementos notacionales de BPMN, respectivamente. Como resultado, obtenemos un conjunto de términos de proceso CSP+T detallados (es decir, el sistema o el BPTM) que describen completamente el comportamiento temporal del sistema o del BPTM. En este sentido, la verificación que se realiza se refiere al comportamiento de los componentes del sistema o del BPTM derivado del modelo del BP expresado a través de procesos CSP+T que especifican ese comportamiento; es decir, la composición de los términos de proceso CSP+T que representa los comportamientos individuales de los componentes del sistema o de los participantes del BPTM.

A través de los formalismos que conforman *Especificar el sistema o BPTM* (véase parte media–izquierda de la Fig. 4.2 (a) y parte inferior–central de la Fig. 4.2 (b), respectivamente), algunos requisitos no–funcionales (es decir, libre de bloqueo, fiabilidad) y restricciones temporales (es de-

cir, puntualidad, cumplimiento de plazos) que el sistema o el BPTM deben cumplir son *especificados* con CCTL. A continuación, estas propiedades se expresan mediante un conjunto de términos de proceso CSP+T que representan el comportamiento abstracto esperado para el sistema o para el BPTM. Como resultado, obtenemos un conjunto de términos de proceso CSP+T que especifican los aspectos de comportamiento y restricciones temporales de los componentes que implementan un sistema o de los participantes involucrados en la realización de un BPTM.

Una vez obtenido el modelo del sistema o BPTM, y el comportamiento esperado para el sistema o BPTM, respectivamente, podemos proceder a llevar a cabo la *Verificación de la corrección del sistema o BPTM* (véase parte inferior de la Fig. 4.2 (a) y parte derecha de la Fig. 4.2 (b), respectivamente) de acuerdo a las reglas de los cálculos de proceso basados en CSP. Por el Teorema 4.1 obtenemos la verificación completa del comportamiento del sistema o BPTM, de acuerdo a la relación (4.1). La aplicación práctica de la relación anterior incluye la realización de un proceso inductivo de 'chequeo de satisfacción' sobre el alcance de la cantidad ($i : 1, \ldots, n$) de componentes del sistema o participantes del BPTM. Usamos la herramienta FDR2 [71] para automatizar esta prueba.

4.5.2 Proceso de verificación

El EFVC consta de los siguientes procesos integrados de acuerdo a la técnica de MC y la teoría de autómatas (ver Fig. 4.3):

Interpretación del sistema o BPTM. Como resultado de la aplicación de la actividad *Modelar y especificar el comportamiento del sistema o BPTM*, se obtiene la descripción completa del comportamiento del sistema o del BPTM (ver parte central derecha de la Fig. 4.3). Modelado por el término de procesos CSP+T $T(C)$, el sistema o BPTM es *interpretado* a través de un conjunto de términos de proceso CSP+T $T(C_i)$ usando MEDISTAM–RT (para el caso de un sistema) o aplicando las reglas de transformación descritas en la sección 3.4.3 (para el caso de BP modelados con BPMN), respectivamente. De esta forma, se obtiene la definición sintáctica que especifica cómo el comportamiento de los componentes del sistema o de los participantes del BPTM es llevado a cabo en tiempo de ejecución, de acuerdo a la secuencia discreta de eventos en los cuales los componentes del sistema o los participantes del BPTM, respectivamente, están involucrados. El artefacto [*Comportamiento de la realización del sistema o del BPTM en términos de CSP+T*] conforma el modelo del sistema o el BPTM.

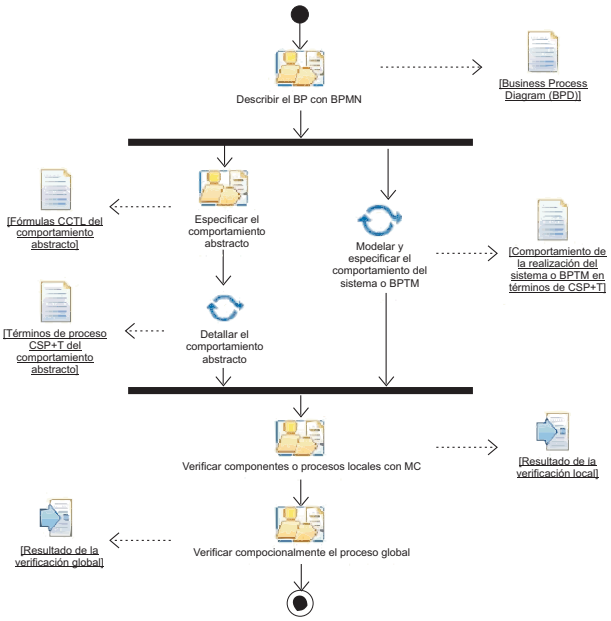


Fig. 4.3 Diagrama de actividades del EFVC.

Especificación de propiedades. En paralelo a lo anterior, los requisitos y las restricciones temporales que el sistema o el BPTM debe cumplir son *especificados* a través de fórmulas CCTL (ver parte central izquierda de la Fig. 4.3). De esta manera obtenemos el artefacto [*Fórmulas CCTL del comportamiento abstracto*] como resultado de la actividad *Especificar el comportamiento abstracto*. Posteriormente, estas propiedades son expresadas por los términos de proceso CSP+T $T(\phi_i)$, $T(\psi_i)$ y $T(\neg\delta)$, siguiendo las dos fases descritas en la sección 4.2, relativas a las propiedades del sistema o BP. Estas fases hacen posible la actividad *Detallar el comportamiento abstracto*, obteniendo como resultado el artefacto [*Términos de proceso CSP+T del comportamiento abstracto*] (ver parte central izquierda de la Fig. 4.3). Como consecuencia, expresamos las propiedades del sistema o del BPTM en el mismo lenguaje de especificación que el modelo del sistema o del BPTM, pudiendo razonar y ejecutar el MC en un mismo dominio semántico.

Verificación. Finalmente, obtenidos los resultados de los procesos anteriores realizados en paralelo, procedemos a la verificación del comportamiento del sistema o del BPTM, ejecutando los siguientes pasos:

1. En primer lugar, ejecutando la actividad *Verificar componentes o procesos locales con MC*, los procesos locales $T(C_i)$ son *verificados automáticamente* contra el conjunto de términos de proceso $T(\phi_i)$,

$T(\psi_i)$ y $T(\neg\delta)$ (ver parte inferior de la Fig. 4.3). De acuerdo con las semánticas de trazas y de fallos de los cálculos basados en CSP, se procede a verificar:

$$T(\phi_i) \sqsubseteq_T T(C_i) \wedge T(\psi_i) \sqsubseteq_T T(C_i) \wedge T(\neg\delta) \sqsubseteq_T T(C_i) \quad (4.11)$$

$$T(\phi_i) \sqsubseteq_F T(C_i) \wedge T(\psi_i) \sqsubseteq_F T(C_i) \wedge T(\neg\delta) \sqsubseteq_F T(C_i) \quad (4.12)$$

Para verificar las relaciones anteriores, trabajamos de acuerdo con el modelo semántico de CSP sin operadores temporales, siguiendo el enfoque *timewise refinement* [190, 224] (ver Definiciones 2.14 y 2.15). De esta manera, verificamos las relaciones previas según el modelo no temporizado de CSP y después las usamos en el análisis temporizado de CSP+T. Hacemos esta actividad con la herramienta FDR2 [71] para llevar a cabo la verificación de los procesos locales que representan los componentes de un sistema o los participantes dentro de un BPTM.

2. En segundo lugar, obtenemos la verificación de la corrección del comportamiento de los componentes o participantes $T(C_i)$ y generamos el artefacto [*Resultado de la verificación local*], de acuerdo con las siguientes relaciones:
   - Relativas a aspectos de seguridad:

$$\begin{aligned} &\forall t \in \mathsf{traces}(T(\phi_i)) \exists t' \in \mathsf{traces}(T(C_i)) : t' \Rightarrow \phi_i \Leftrightarrow T(C_i) \models \phi_i \\ &\forall t \in \mathsf{traces}(T(\psi_i)) \exists t' \in \mathsf{traces}(T(C_i)) : t' \Rightarrow \psi_i \Leftrightarrow T(C_i) \models \psi_i \\ &\forall t \in \mathsf{traces}(T(\neg\delta)) \exists t' \in \mathsf{traces}(T(C_i)) : t' \Rightarrow \neg\delta \Leftrightarrow T(C_i) \models \neg\delta \end{aligned} \quad (4.13)$$

   - Relativas a aspectos de vivacidad:

$$\begin{aligned} &\forall (t,X) \in \mathbf{SF}[\![T(\phi_i)]\!] \exists (t',X) \in \mathbf{SF}[\![T(C_i)]\!] : (t',X) \Rightarrow \phi_i \Leftrightarrow T(C_i) \models \phi_i \\ &\forall (t,X) \in \mathbf{SF}[\![T(\psi_i)]\!] \exists (t',X) \in \mathbf{SF}[\![T(C_i)]\!] : (t',X) \Rightarrow \psi_i \Leftrightarrow T(C_i) \models \psi_i \\ &\forall (t,X) \in \mathbf{SF}[\![T(\neg\delta)]\!] \exists (t',X) \in \mathbf{SF}[\![T(C_i)]\!] : (t',X) \Rightarrow \neg\delta \Leftrightarrow T(C_i) \models \neg\delta \end{aligned} \quad (4.14)$$

3. Por último, llevando a cabo la actividad *Verificar composicionalmente el proceso global* (ver parte inferior de la Fig. 4.3) y mediante la aplicación del Teorema 4.1, obtenemos la verificación completa del comportamiento del sistema o del BPTM $T(C)$, de acuerdo a la relación (4.1) instanciada para los términos de proceso CSP+T; es decir, $T(C) = \|_{i:1..n}\ T(C_i)$). Así, obtenemos el artefacto [*Resultado de la verificación global*].

Presentado nuestro enfoque conceptual y la intanciación en el EFVC, en el próximo capítulo expondremos las ideas iniciales para lograr su automatización y el desarrollo de la herramienta que constituye el primer paso hacia el soporte automático de las aportaciones derivadas de nuestro trabajo.

Capítulo 5
Diseño de una herramienta de verificación composicional para procesos de negocio

Resumen El objeto de este capítulo es proponer el diseño general del conjunto de herramientas que sirvan de apoyo de la mayor parte del enfoque de verificación composicional que se describe en este trabajo; así como presentar el diseño específico y la implementación de la herramienta que automatiza la aplicación de las reglas de transformación que nos permite traducir un diagrama BPMN a términos de proceso CSP+T. En la sección 5.1 exponemos la propuesta que describe el diseño arquitectónico a alto nivel del conjunto de herramientas. Luego, en la sección 5.2, presentamos el diseño detallado de la herramienta de transformación BTRANSFORMER, la cual automatiza las reglas de transformación presentadas en la sección 3.4.3. En la sección 5.3 se hace una breve presentación de la herramienta de transformación obtenida y se muestra su funcionamiento a través de un ejemplo. Finalmente, en la sección 5.4, se indican algunas pruebas preliminares realizadas a BTRANSFORMER.

5.1 Propuesta general del Workbench

En el área de la IS la garantía de que los enfoques, métodos y técnicas, sean utilizados es que éstos estén soportados por herramientas de software. Por esta razón, el EFVC fue definido pensando en su inminente automatización. Soportado por la infraestructura matemática que lo define, tal como fue presentado a lo largo del capítulo 4, el EFVC permite la incorporación de distintas herramientas que implementan varios aspectos del complejo proceso de verificación. En este sentido, la automatización del EFVC puede ser técnicamente definida como un *Workbench* [74].

Un Workbench es *un conjunto de herramientas de software que soportan una fase concreta del proceso software, que funcionan colaborativamente para proporcionar un apoyo integral* [74, 172, 196]. Además, cuenta con servicios comunes que son utilizados por todas las herramientas y contempla el soporte para la integración de datos [172, 196]. El diagrama de paquetes de UML que se presenta en la Fig. 5.1 muestra la propuesta general de automatización del EFVC mediante un Workbench cuyo objetivo es *soportar la verificación composicional sobre la base del uso de los lenguajes y formalis-*

mos presentados a lo largo de este trabajo. Se estima que la implementación completa del Workbench se logre con el trabajo posterior relacionado con la presente tesis doctoral.

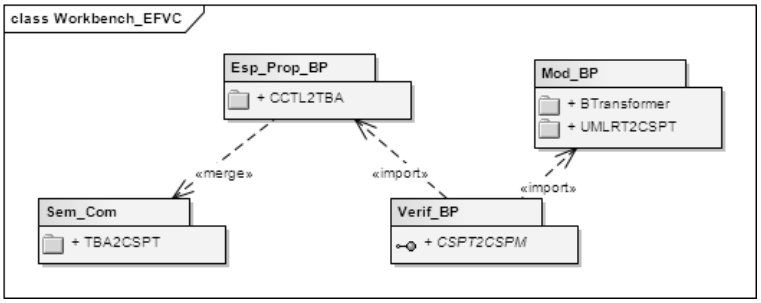


Fig. 5.1 Diagrama de paquetes del Workbench para el soporte automatizado del EFVC.

En la Fig. 5.1 se presentan los 4 paquetes que consideramos constituyen la estructura general del Workbench. Dado que el EFVC ha sido concebido como un enfoque de verificación composicional abierto y extensible, admite el uso de: (1) diversas notaciones para el modelado de BPs, (2) distintas lógicas de especificación de propiedades, y (3) diferentes cálculos de procesos que permiten realizar la verificación composicional. En este sentido cada paquete contendrá finalmente el conjunto de herramientas que se vayan adicionando para implementar las distintas combinaciones que se pueden generar con los elementos anteriores (1 a 3). Por ello, la estructura de la arquitectura que proponemos para el Workbench queda como sigue:

Mod_BP (Modelo del BP). Contiene las herramientas que permiten obtener la especificación formal utilizando algún cálculo de procesos del modelo del BP. El modelo se expresa inicialmente con alguna notación semi–formal de uso común en BPM. Las herramientas software incluidas aquí implementan reglas de transformación de notaciones no formales o semi–formales a especificaciones formales del BPTM que pueden ser posteriormente verificados por herramientas de verificación automática existentes en el mercado.

Esp_Prop_BP (Especificación de propiedades del BP). Incluye a las herramientas que se utilizan para la transformación de las propiedades especificadas de un modelo del BP en alguna lógica temporal a algún formalismo intermedio (p.e., TBA) que permita su posterior transformación al mismo dominio semántico del modelo del BP.

Sem_Com (Semántica común). Conforma el grupo de herramientas que soportan la transformación de algún formalismo intermedio (p.e., TBA) al cálculo de procesos en el cual está especificado el modelo del BP. Este

paquete proporciona la posibilidad de que las propiedades del modelo del BP y el modelo del BP se especifican en un mismo dominio semantico.

Verif_BP (Verificación del BP). Agrupa las herramientas que traducen modelos formales objeto de verificación al lenguaje de descrpción formal (del sistema) que requieren los comprobadores de modelos como entrada para llevar a cabo la verificación. Este conjunto de herramientas constituye la *interfaz* entre los modelos formales obtenidos por la aplicación del EFVC y los comprobadores de modelos disponibles en el mercado.

Sobre la base de lo anterior y de acuerdo a la Fig. 5.1, hemos denominado BTRANSFORMER a la herramienta que automatiza la aplicación de la semántica formal temporal que proponemos en la sección 3.3 para los elementos notacionales de tiempo de BPMN, mientras que las reglas de transformación de UML–RT a CSP+T que se definen en MEDISTAM–RT [21] se implementarían en la herramienta *UMLRT2CSPT*. Por su parte, la herramienta *CCTL2TBA* implementaría el algoritmo para obtener los TBAs a partir de las fórmulas CCTL que se presentó en la sección 3.5.1, y la herramienta *TBA2CSPT* implementaría el procedimiento descrito en [138] que permite obtener los términos de proceso CSP+T a partir de los TBAs. Por último, la herramienta *CSPT2CSPM* sería un traductor entre el lenguaje de especificación CSP+T y el lenguaje de programación CSP_M[1], que actuaría como interfaz entre las herramientas *TBA2CSPT*, BTRANSFORMER y *UMLRT2CSPT* y el comprobador de modelos FDR2 [71] para lograr la verificación de los componentes de un SCS o de los procesos locales de un BP. En síntesis, el uso de este conjunto de herramientas, integradas a través de XML para el intercambio de datos, constituyen el Workbench que automatiza el uso del EFVC aprovechando las fortalezas del cálculo de procesos CSP+T.

Con la finalidad de mostrar la viabilidad de la arquitectura general propuesta para automatizar el EFVC (ver Fig. 5.1), en la próxima sección se presentan los elementos más significativos del diseño de la herramienta BTRANSFORMER. Esta herramienta constituye un primer paso hacia la automatización del EFVC y fue seleccionada por su representatividad como una aplicación de un Lenguaje de Transformación de Modelos (*Model Transformation Language*, MTL).

[1] Lenguaje de programación basado en CSP utilizado por FDR2 [71].

5.2 Herramienta BTRANSFORMER

Esta herramienta ha sido concebida como un gestor de transformaciones entre modelos, ya que lo que logramos a través de ella es la transformación de un modelo BPMN a un modelo CSP+T que especifica el comportamiento del BP y que nos permite su posterior verificación. Por ello, la implementación propuesta para esta herramienta integra las facilidades que vienen proporcionadas por un MTL[2], buscando que sea familiar para el analista y a partir de la cual se puedan generar automáticamente especificaciones formales en CSP+T de un modelo BPMN.

Cabe indicar en este momento que el diseño aquí presentado podría servir de base para el futuro desarrollo de la herramienta *UMLRT2CSPT*, ya que ambas tienen la finalidad de soportar la aplicación de reglas de transformación para obtener modelos formales en CSP+T. Sin embargo, esto dependerá fundamentalmente del análisis de las características de los modelos a transformar y de las características del MTL.

Para el diseño de la herramienta BTRANSFORMER fue decisivo el encontrar un entorno de trabajo natural para los usuarios de BPMN al cual fuera factible integrarle una opción que permitiese generar una especificación en CSP+T. Esto lo conseguimos a través de la plataforma Eclipse[3], ya que la misma cuenta con el editor Intalio[4]. Lo cual aumentó la factibilidad del proyecto debido a que no fue necesario procesar el modelo BPMN al estar disponibles los servicios del editor de BPMN[5] en esta plataforma.

Por la filosofía de Eclipse de ser una plataforma de programación abierta que permite la creación y mantenimiento de Entornos Integrados de Desarrollo (*Integrated Development Environment*, IDE), y que las transformaciones que lleva a cabo la herramienta BTRANSFORMER son de modelo a modelo, encontramos que el lenguaje de transformación más adecuado para lograr nuestros propósitos y que está disponible para esta plataforma es el *ATLAS Transformation Language* (ATL).

ATL es una propuesta de ATLAS Group (INRIA & LINA, Universidad de Nantes) y fue desarrollado como parte de la plataforma *ATLAS Model Management Architecture* (AMMA) [109]. ATL comparte las características comunes y el mismo conjunto de requerimientos definidos en *Query/View/Transformation* (QVT) *Request For Proposal* (RFP) [107]. Es compatible con los estándares de OMG. Por tanto, es posible describir trans-

[2] Un MTL es aquel que se utiliza para definir transformaciones de modelos, especificadas como determinados tipos del metamodelo del lenguaje fuente se convierten en otro tipo del metamodelo del lenguaje objetivo [116].

[3] `http://www.eclipse.org/`

[4] `http://www.intalio.com/`

[5] `http://www.eclipse.org/bpmn/`

formaciones modelo a modelo —ambos deben ser instancias del *Meta Object Facility* (MOF)— y permite definir pre y pos condiciones en un lenguaje ya conocido: *Object Constraint Language* (OCL) [107]. En la Fig. 5.2 se indica el enfoque transformacional de ATL de forma gráfica, el cual es descrito en detalle en [108].

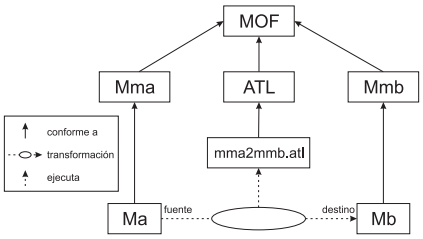


Fig. 5.2 Vista general del enfoque transformacional de ATL.

Tal como muestra la Fig. 5.2 [108], ATL permite aplicar el patrón de transformación en el cual un modelo fuente Ma se transforma en un modelo objetivo Mb, y la transformación es especificada mediante una definición o programa (mma2mmb.atl) [108]. Los modelos fuente y objetivo están conformes a los metamodelos MMa y MMb. A su vez los metamodelos están conformes al metamodelo MOF [109].

5.2.1 Revisión de requisitos

5.2.1.1 Funcionales

En la Tabla 5.1 se muestra el *Requisito Funcional* (RF) que cumple la versión de la herramienta de transformación BTRANSFORMER. Este RF constituye la característica principal de dicha herramienta.

5.2.1.2 No Funcionales

En la Tabla 5.2 se muestran los *Requisitos No Funcionales* (RNF) que cumple la versión de la herramienta que aquí se describe. Estos requisitos constituyen las características suplementarias de la herramienta.

Definidos y analizados en RF y los RNF que ha de cumplir la herramienta de transformación BTRANSFORMER, se presenta en la siguiente sección un análisis de los Casos de Uso (CUs) considerados en esta versión.

I.D.	Atributos	
RF.1	Nombre	Aplicación de reglas de transformación desde modelos BPMN a esecificaciones CSP+T.
	Descripción	La herramienta debe permitir la aplicación de las reglas de transformación asociadas al tipo de transformación BPMN a CSP+T.
	Tipo	Funcional
	Detalles y Restricciones	La herramienta debe leer el archivo con el modelo BPMN fuente y generar el archivo con el modelo CSP+T objetivo, utilizando las reglas de transformación existentes para este tipo de transformación.
	Versión	1.0
	Condición	Debe (obligatorio).

Tabla 5.1 Requisito Funcional para la herramienta BTRANSFORMER.

I.D.	Atributos	
RNF.1	Nombre	Estándar de intercambio de datos.
	Descripción	La herramienta debe seguir la especificación del lenguaje XML para el intercambio de datos.
	Tipo	No Funcional
	Detalles y Restricciones	La herramienta debe leer el archivo XML que contenga el modelo BPMN fuente y generar el archivo XML que contenga el modelo CSP+T objetivo de una transformación.
	Versión	1.0
	Condición	Debe (obligatorio).
RNF.2	Nombre	Portabilidad.
	Descripción	La herramienta debe desarrollarse bajo la plataforma Eclipse y siguiendo los lineamientos de la Máquina Virtual de Java.
	Tipo	No Funcional
	Detalles y Restricciones	Máquina Virtual de Java nivel 1.3 o superior (por ejemplo, *Java Development Kit* (JDK) 1.3 o superior). Eclipse *Software Development Kit* (SDK) 3.5 o superior para la plataforma de ejecución.
	Versión	1.0
	Condición	Debe (obligatorio).

Tabla 5.2 Requisitos No Funcionales para la herramienta BTRANSFORMER.

5.2.2 Modelo de Casos de Uso

La versión actual de la herramienta de transformación cuenta con un CU, tal como se describe brevemente en la Tabla 5.3.

I.D.	Actor	Nombre del Caso de Uso (CU)	Descripción del CU
CU.1	Analista	Generar transformación.	Permite que el Analista genere el modelo CSP+T a partir de un modelo BPMN fuente, aplicando las reglas de transformación BPMN a CSP+T.

Tabla 5.3 Descripción de los CUs de la herramienta BTRANSFORMER.

A continuación, en la Fig. 5.3, se muestra el diagrama de CUs de la herramienta de transformación.

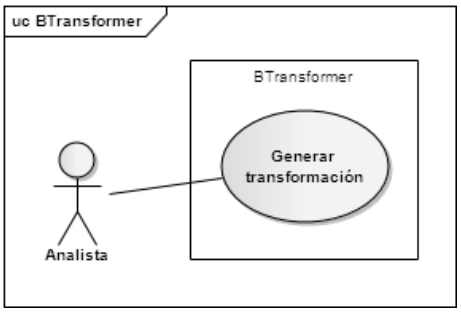


Fig. 5.3 Diagrama de CUs de la herramienta BTRANSFORMER.

En la Tabla 5.4 se presentan los detalles del CU *Generar transformación* que contempla la versión de la herramienta BTRANSFORMER diseñada en este trabajo.

5.2.3 Vista lógica

Con esta vista se exponen las abstracciones principales del diseño de la herramienta, en su mayoría sugeridas por el CU y los requisitos especificados con anterioridad. Antes de mostrar los diagramas más importantes de esta vista, se expondrán algunas características de ATL que tiene relación directa con el diseño de la herramienta.

El lenguaje ATL usa construcciones de tipo *declarativo* e *imperativo*. Las transformación definidas son unidireccionales, operan sobre modelos fuentes sólo–lectura, y producen un modelo objetivo sólo–escritura. Para que la transformación sea bidireccional es necesario definir una transformación para cada sentido [108, 107].

Las transformaciones en ATL se definen en forma de *módulos*. Cada módulo debe tener una sección de *encabezado*, un número de *métodos* (*helpers*) y un número de *reglas* (*rules*) [108, 107]. En el encabezado se declaran el modelo objetivo y el modelo fuente. Los métodos son construcciones que permiten realizar operaciones sobre los modelos fuente y/u objetivo, o declarar atributos dentro de la transformación [109]. La reglas de ATL de tipo declarativo se denominan *matched rules*, y están compuestas de dos patrones: *fuente* y *objetivo*. Se puede especificar que las reglas sean explícitamente invocadas por otras y que se ejecuten una única vez. ATL soporta la herencia entre reglas. Estas reglas pueden presentar opciones de estilo imperativo; es decir, es posible invocar llamadas a otras operaciones,

I.D. del CU: CU.1	**Nombre del CU:** Generar transformación.
Descripción: Permite que el Analista genere el modelo CSP+T a partir de un modelo BPMN fuente, aplicando las reglas de transformación BPMN a CSP+T.	
Tipo: Básico.	
Requerimientos: RF.1.	
Precondición: La plataforma Eclipse SDK 3.5 o superior ha sido activada. Las reglas de transformación BPMN a CSP+T existen en la herramienta. El archivo que contiene el modelo BPMN tiene una ruta válida.	
FLUJO BÁSICO	
Actor	**Herramienta**
1) Se inicia cuando el Analista solicita generar un modelo CSP+T a partir de un modelo BPMN, seleccionando la sub–opción BPMN To CSPT de la opción Transformation del menú contextual que se muestra al pulsar el botón derecho del ratón sobre el archivo XML que contiene el modelo BPMN.	2) Solicita confirmación de la aplicación de las reglas de transformación BPMN a CSP+T.
3) Confirma la aplicación del tipo de transformación pulsando la opción *OK*.	4.1) Aplica las reglas de transformación BPMN a CSP+T. 4.2) Guarda la transformación realizada. 4.3) Registra la instancia de la transformación realizada. 4.4) Indica al Usuario la ruta de acceso del archivo generado que contiene el modelo CSP+T objetivo.
5) Finaliza cuando el Analista pulsa la opción *Salir*.	6) Cierra la herramienta.
FLUJOS ALTERNOS	
Actor	**Herramienta**
A7) Este CU se cancela cuando el Usuario selecciona la opción *Salir* en cualquier momento.	8) Cierra la herramienta.
Poscondición: Un archivo que contiene el modelo CSP+T objetivo ha sido generado y está disponible en una ruta válida. Una nueva instancia de la transformación BPMN a CSP+T ha sido registrada en la herramienta.	
Requisito especial: No tiene.	
Puntos de extensión: No tiene.	

Tabla 5.4 Especificación del CU Generar transformación.

tomar el control de la ejecución de la transformación y extraer datos de los modelos para ser utilizados en el modelo objetivo [107, 108].

La vista lógica está constituida principalmente por el Diagrama conceptual y el Diagrama de clases de la herramienta, los cuales se presentan en las Figs. 5.4 y 5.5, respectivamente.

La Fig. 5.4 muestra los conceptos relacionados al proceso de generación de un modelo CSP+T objetivo a partir de un modelo BPMN fuente usando reglas de transformación descritas la sección 3.4.3. La herramienta permite generar la especificación en CSP de un modelo BPMN estándar, y en el caso

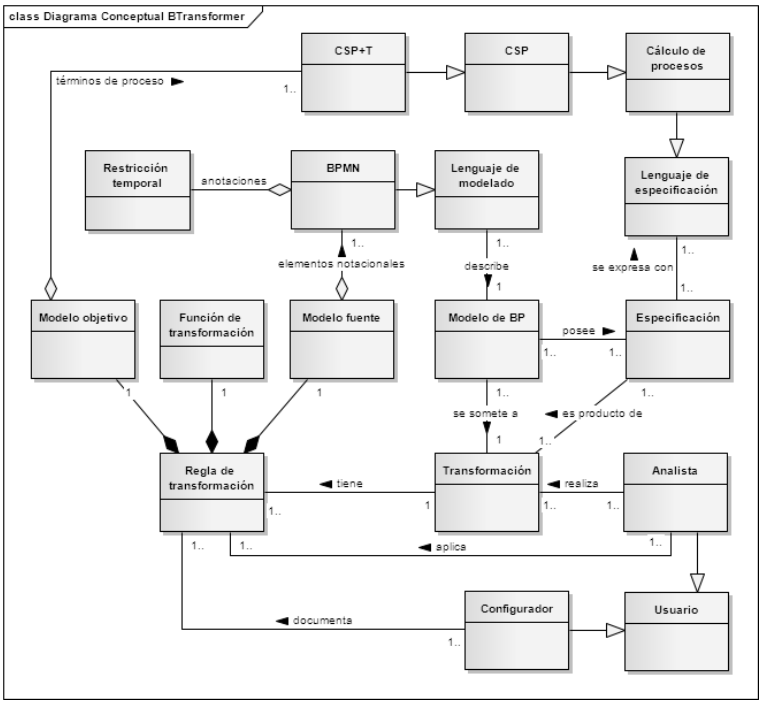


Fig. 5.4 Diagrama conceptual de la herramienta BTRANSFORMER.

de que se agreguen al modelo BPMN las especificaciones de tiempo (ver Apéndice E), se genera también la especificación en CSP+T.

Tomando en cuenta los conceptos relacionados en la Fig. 5.4, se definieron las clases (ver Fig. 5.5) que se implementaron durante la construcción del *plug–in*[6] de la herramienta de transformación y que permitió satisfacer el RNF.2 relacionado con la portabilidad de la herramienta. Todas estas clases presentan abstracciones de la herramienta de transformación. La Fig. 5.5 muestra el diagrama de clases de la herramienta de transformación.

La clase AccionBPMN2CSPT en la acción que implementa el menú que permite acceder a la funcionalidad desde el entorno de Eclipse. La Clase AccionAplicarTransformacion hace uso de los servicios del entorno ATL en base a los valores de una o más clases de Configuracion. En la clase Latex se le da formato a las instrucciones resultante de la transformación, se evalúan los tiempos. La clase LectorEscritor permite manejar la persistencia al guardar las instrucciones en un archivo.

[6] Un *plug–in* es la unidad de funcionalidad más pequeña de la plataforma Eclipse que puede ser distribuida de manera separada.

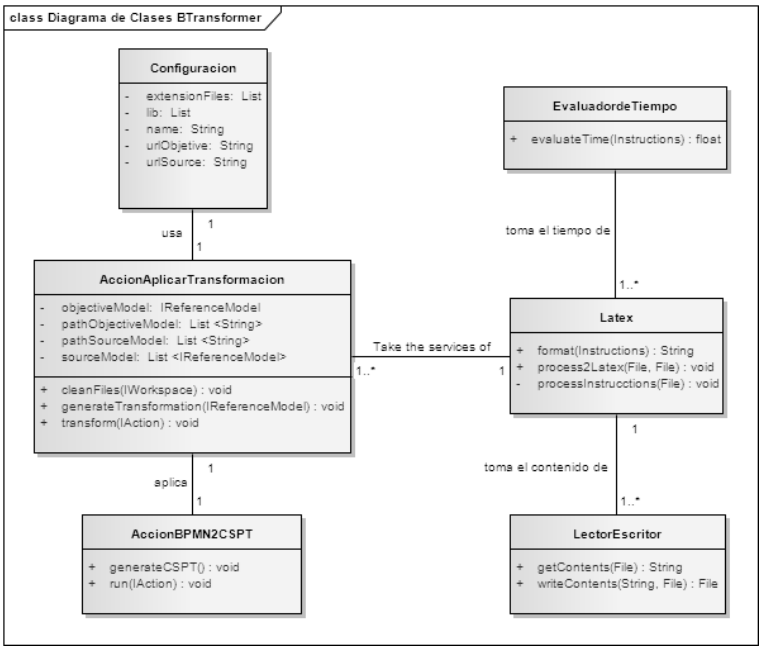


Fig. 5.5 Diagrama de clases de la herramienta BTRANSFORMER.

En la Fig. 5.6 se muestra el diagrama de secuencia que detalla cómo dinámicamente las clases especificadas en el diagrama de clases anterior (Fig. 5.5) interactúan para lograr la realización del CU *Generar transformación* que implementa la versión actual de la herramienta.

5.2.4 Vista física o de implementación

La vista física de la arquitectura de la herramienta BTRANSFORMER muestra los componentes del sistema. Considerando que la plataforma Eclipse sigue una filosofía basada en componentes, la Fig. 5.7 muestra la interrelación entre los diferentes componentes que implementan la arquitectura de la herramienta de transformación, donde es posible visualizar en el diseño la integración con el editor Intalio en la plataforma Eclipse. Tal como se indicó anteriormente, al estar disponibles los servicios del editor de BPMN no fue necesario desarrollar el procesamiento del modelo BPMN, lo cual facilitó la implementación de la herramienta.

Usualmente, una herramienta pequeña es diseñada como un solo plug–in, mientras que una herramienta compleja distribuye su funcionalidad a lo

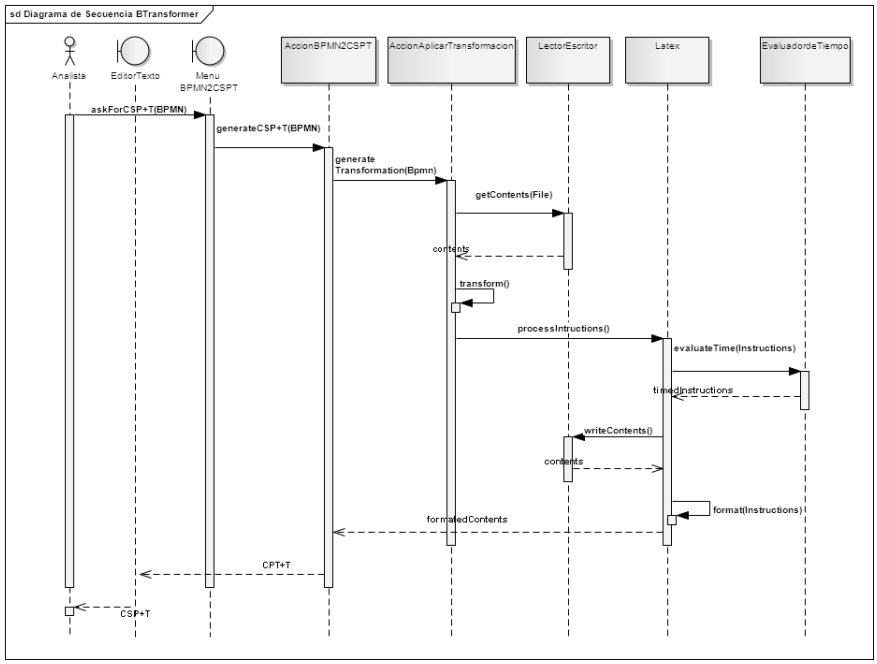


Fig. 5.6 Diagrama de secuencia del CU Generar transformación.

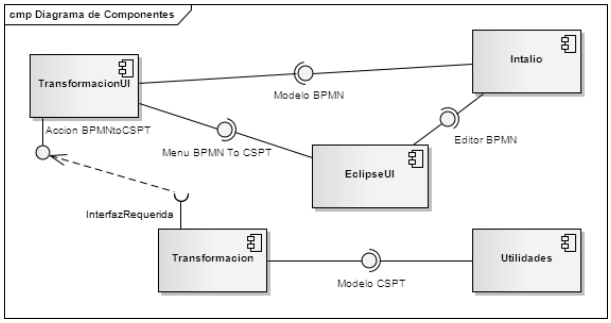


Fig. 5.7 Diagrama de componentes de la herramienta BTRANSFORMER.

largo de varios plug–ins. Los plug–ins son implementados en el lenguaje Java[7], a excepción de un pequeño núcleo (en inglés, *kernel*). Un plug–in típico consiste de código Java en una librería JAR (Java ARchive; en inglés, archivo Java)), algunos archivos de sólo lectura y otros recursos, tal como imágenes plantillas Web, descriptores de mensajes, librerías de código nativo, entre otros. Algunos plug–ins no contienen sólo código. También hay

[7] `http://www.java.com`.

un mecanismo que permite a un plug–in sincronizarse a partir de muchos fragmentos separados, cada uno en su propio directorio [179].

Mediante la acción desde la plataforma Eclipse que invoca el plug–in TransformacionUI (ver Fig. 5.7), se implementa el acceso a la funcionalidad de la herramienta BTRANSFORMER. Utilizando los servicios del plug–in Intalio, el cual controla todo el proceso de modelado BPMN del modelo fuente (incluyendo las anotaciones temporales descritas en el Apéndice E), son aplicadas las reglas de transformación definidas en el plug–in Transformacion. Por su parte, el plug–in Utilidades se encarga de la lectura de las entidades de modelado BPMN del modelo fuente y de la escritura de los procesos CSP+T del modelo objeto. Además de los plug–ins desarrollados, los cuales están basados en la plataforma, el diagrama de secuencia de la Fig. 5.6 incluye los elementos que permiten la implementación de su interfaz .

La descripción de los componentes se encuentra en la Tabla 5.5.

La Tabla 5.5 especifica cómo están constituidos los componentes desarrollados para la herramienta de transformación, definiendo los paquetes y las clases que manejan cada paquete.

Plug–in	Paquete	Clases (.java) o recursos
Utilidades	ve.usb.ve.lisi.utils	LatexATL.java LectorEscritor.java
TransformacionUI	ve.usb.ve.lisi.actions	TransformacionAction.java BPMN2CSPActions.java
	ve.usb.ve.lisi.configuracion	Configuracion.java
Transformacion	ve.usb.ve.lisi.transformacion	EjecutorTransformacion.java
	ve.usb.ve.lisi.metamodelos	Bpmn metamodel.bpmn cspT metamodel.cspt latex metamodel.latex
	ve.usb.ve.lisi.plantillas	Encabezado.txt Fin.txt
	ve.usb.ve.lisi.definiciones	BPMNToInstructions.atl Actividades.atl Eventos.atl Consultas.atl Subprocesos.atl

Tabla 5.5 Paquetes y clases (.java) o recursos, utilizadas para la herramienta BTRANSFORMER.

5.2.5 Vista de implantación

Bajo la plataforma Eclipse la funcionalidad que se provee por medio de los plug–ins debe ser instalada en el mismo computador donde se eje-

cuta Eclipse. El proceso consiste en descomprimir el contenido del plug–in BTRANSFORMER bajo la carpeta *plugins* del directorio de instalación de la plataforma. A partir de ese momento, la nueva funcionalidad es reconocida y puede ser utilizada.

Con relación a los requerimientos de hardware recomendados para una ejecución óptima para la herramienta, se recomienda el uso de:

- Procesador Intel Pentium IV o compatible.
- 1 GB RAM.
- 2 GB libres en Disco Duro.

Los requerimientos mínimos necesarios de software para el uso de la herramienta de transformación BTRANSFORMER:

- Máquina Virtual de Java nivel 1.3 o superior (por ejemplo, JDK 1.3 o superior).
- Eclipse SDK 3.5 o superior para la plataforma de ejecución (Windows XP, Windows® 2000, Windows® 98, Windows® ME, Red Hat Linux Version 7.1 (x86/Motif and x86/GTK)), SuSE Linux 7.1 (x86/Motif and x86/GTK), o Solaris® 8 (SPARC/Motif)).

5.3 Presentación y uso de la herramienta

Después que fue instalado el software necesario para el entorno de desarrollo, la funcionalidad de la herramienta fue contruida y todos los componentes y el CU propuesto fueron implementados. Adicionalmente, algunos casos de prueba fueron aplicados al CU *Generar transformación* (ver sección 5.4). El principal artefacto obtenido fue la versión 1.0 de BTRANSFORMER, incluyendo el código fuente de toda las clases implementadas.

Para ejemplificar el uso de la herramienta de transformación BTRANSFORMER, vamos a mostrar el resultado de la generación del modelo CSP+T que se obtiene a partir del modelo BPMN mostrado en la Fig. 5.8, el cual es descrito en detalle en [13].

La Fig. 5.9 muestra una captura de pantalla que corresponde con la invocación de la herramienta BTRANSFORMER a través de la sub–opción BPMN To CSPT (que activa el plug–in TransformacionUI) de la opción Transformations del menú contextual que se despliega pulsando el botón derecho del ratón sobre el nombre del archivo XML del diagrama BPMN construido con Intalio.

En la Fig. 5.10 se muestran el conjunto de comandos LaTeX generados como resultado de la transformación.

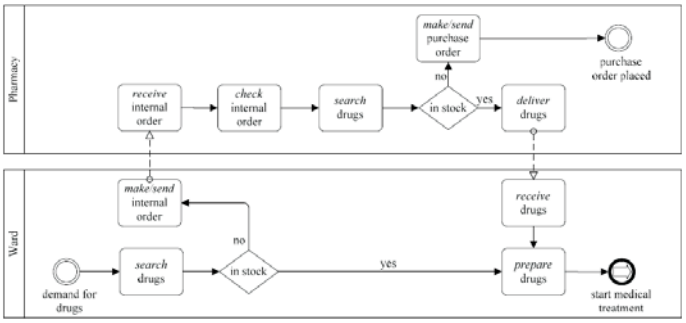


Fig. 5.8 Ejemplo de un proceso logístico.

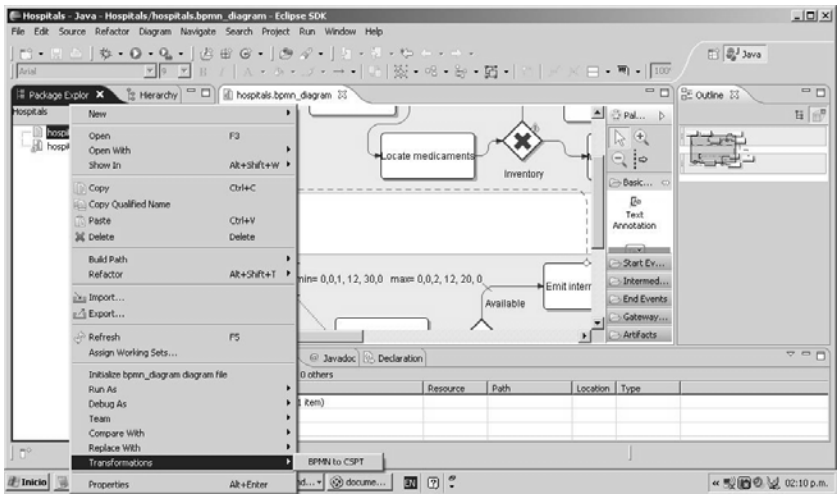


Fig. 5.9 Captura de pantalla de la invocación de BTRANSFORMER.

5.4 Algunas pruebas preliminares

Aunque en el futuro podrían detectarse posibles inconsistencias en la semántica de las especificaciones generadas por la herramienta BTRANSFORMER, al introducirlas en herramientas especializadas como FDR2, actualmente es posible realizar pruebas sintácticas sobre algunos modelos BPMN disponibles. En esta sección se describen los resultados de transformar algunos 'casos de prueba' seleccionados de terceros autores con la finalidad de evitar su modelado con BPMN y que son de acceso público.

Los modelos BPMN utilizados para evaluar el funcionamiento de la herramienta son:

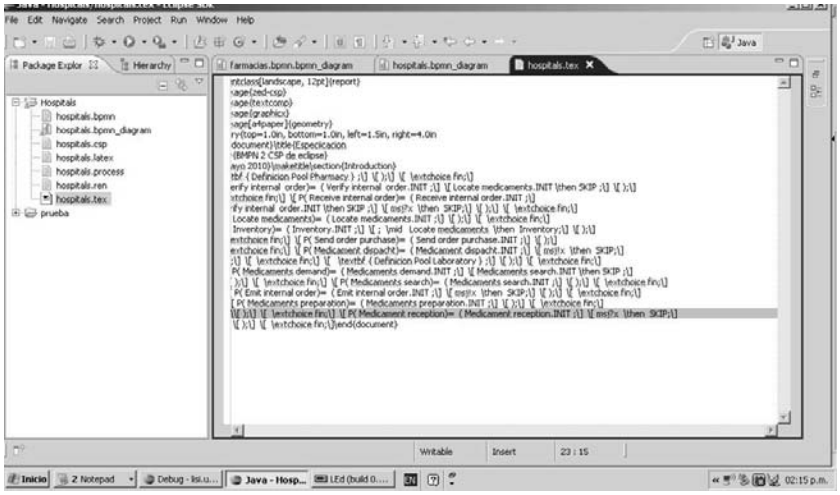


Fig. 5.10 Captura de pantalla del resultado generado por BTRANSFORMER.

- *Cadena de transporte* [117]. Este diagrama representa una cadena de logística con diferentes medios de transporte (terrestre, ferroviario y marítimo). Se modelan el terminal terrestre y el terminal marítimo.
- *Proceso de logística en hospitales* [13]. En este diagrama se describen las actividades necesarias para mantener el inventario de medicamentos en un hospital.

Se definieron las siguientes heurísticas o criterios con la finalidad de detectar inconsistencias en la sintaxis de los procesos CSP+T del modelo objeto generado, con respecto a los elementos notacionales BPMN del modelo fuente:

1. *Completitud de elementos*. todos los elementos de los diagramas de BPMN aparecen reflejados en la especificación semánticamente equivalente en CSP+T.
2. *Número de procesos*. Existe al menos un proceso por actividad definida en el diagrama.
3. *Completitud de relaciones*. Es posible establecer relaciones entre dos procesos siempre que se presente una relación entre dos o más actividades en el modelo.
4. *Seguridad comportamental*. El conjunto de secuencias de ejecución de los procesos en la especificación ha de estar incluido en (o coincidir con) el conjunto de secuencias del modelo. Es decir, hay que asegurar que la especificación en CSP+T no 'invente' secuencias de ejecución que no existen en el modelo BPMN.

En la Tabla 5.6 se presentan los resultados de aplicar los criterios indicados anteriormente a la transformación obtenida por el uso de BTRANSFORMER.

Modelo	Completitud de elementos	Número de procesos	Completitud de relaciones	Seguridad comportamental
Cadena de transporte	Si	12	Si	Si
Logística en hospitales	Si	22	Si	Si

Tabla 5.6 Resultados de pruebas preliminares de BTRANSFORMER.

De acuerdo a los resultados de la Tabla 5.6, se puede afirmar que en ambos casos la especificación generada fue completa, y que se encontraron el mínimo de definiciones de procesos esperados. También se reflejaron las relaciones en la especificación y que se preserva la seguridad comportamental.

Expuesto el desarrollo de la primera herramienta que representa el esfuerzo inicial hacia la automatización del EFCV, en el próximo capítulo se describen dos casos de interés empresarial e industrial que muestran la aplicación y viabilidad del EFVC.

APLICACIÓN

Capítulo 6
Casos de ejemplo

Resumen El objeto de este capítulo es presentar la aplicación de las aportaciones de la presente tesis doctoral a dos casos reales. El primero, explicado en la sección 6.1, se refiere a un BP considerado crítico para la estrategia de Gestión de las Relaciones con el Cliente (*Customer Relationship Management*, CRM). El segundo, desarrollado en la sección 6.2, corresponde al sistema que implementa el proceso crítico del protocolo de comunicación *Distributed Data Base Manager* (DDBM) que constituye el corazón de una red de comunicación de teléfonos móviles.

6.1 Gestión de las Relaciones con el Cliente

CRM es una estrategia que permite lograr y mantener la relación entre una empresa y sus clientes, buscando un trato con cada cliente por separado, ofreciéndoles una atención individual y a su vez, participación en la relación comercial [30, 40, 144]. Es una estrategia de negocios que busca seleccionar y gestionar a los clientes y así optimizar su valor a largo plazo [202]. Por otra parte, en [198] se define CRM como una estrategia de negocios que nos proporciona la tecnología actual con la que las organizaciones aumentan el conocimiento del cliente para construir relaciones rentables, basadas en la optimización del valor que éstas le ofrecen a los clientes y que reciben también de ellos. Requiere una filosofía de negocios basada en el cliente y una cultura para soportar efectivamente los procesos comerciales, ventas y servicios. Las aplicaciones CRM permiten la gerencia efectiva de las relaciones con los clientes, ayudando a que una empresa tenga el liderazgo, la estrategia y los hábitos adecuados [144].

El nuevo paradigma de CRM refleja un cambio en las relaciones comerciales tradicionales que se describe como *gerencia del cliente*. Este concepto supera de alguna manera la tendencia de las organizaciones a captar nuevos clientes. Aparecen como primer elemento a considerar ¿cómo mantener los clientes?; es decir, ¿cómo hacer que la relación con el cliente perdure en el tiempo?. Para lograr esto, se requiere información relacionada con los patrones de compra de los clientes y preferencias de productos, entre otros. Más que la automatización de las ventas, CRM requiere la habilidad de otor-

garle más importancia a los clientes claves y a los datos corporativos para emprender mejores decisiones de negocios [165, 176, 180]. Para lograr este objetivo existen una serie de aspectos relacionados [40, 144, 202]:

Los Procesos. Los clientes se relacionan con la organización a través de estos. Para [202], los BP claves son: Mercadeo, Ventas y Servicios. Además de estos procesos pueden existir, dependiendo del área de negocio de la empresa, procesos que se ven afectados de manera directa y que también deben ser considerados.

El Factor Humano. Las personas tienen un papel clave dentro de la estrategia de CRM, tanto en lo referido a los empleados dentro de la organización, quienes deben estar inmersos en un cambio de hábitos comerciales, como los clientes.

Las TIC. Son el elemento facilitador de la implementación de la estrategia de CRM, lo cual hace necesario conocer cuáles son estas tecnologías y cómo favorecen la estrategia de CRM.

Los objetivos adicionales que colaboran con el logro del objetivo principal del CRM son [144]: (1) maximizar la información del cliente; (2) identificar nuevas oportunidades de negocio; (3) mejorar el servicio al cliente; (4) procesos optimizados y personalizados; (5) mejora de ofertas y reducción de costos; (5) identificar los clientes potenciales que mayor beneficio generen para la empresa; (6) fidelizar al cliente, aumentando las tasas de retención de clientes; y (7) aumentar la cuota de gasto de los clientes.

CRM es, finalmente, la compleja combinación de negocio y factores técnicos que deberían estar alineados con una estrategia comercial [30].

6.1.1 Procesos de negocio involucrados

El mercado para el negocio CRM se compone principalmente de los clientes reales y potenciales de las organizaciones interesadas en estrechar efectivamente la relación estos, brindando para ello los servicios y herramientas adecuadas según el tipo de organización. El general, una organización orientada hacia el CRM cubre principalmente los siguientes BP: (1) Atención a la estrategia CRM; (2) Captación de clientes; (3) Venta de productos/servicios; (4) Personalización del servicio requerido por los clientes; (5) Análisis de competidores; (6) Mantenimiento o retención de clientes; (7) Programas de incentivos o comisiones; y (8) Estudio del patrón de comportamiento de los clientes y de los productos, entre otros. Cada uno de estos procesos puede o no estar presente como parte de los BP de una organización; depende del tipo de negocio y los clientes.

Los BP relacionados con CRM se implantan en base al modelo de negocio y a la valoración de estos, ampliando la solución o limitándola a aspectos muy puntuales del negocio pero claves para su rentabilidad [30]. Por ello, se considera que los BP medulares para el negocio son Producir/Prestar Servicio, Vender Producto/Servicio y Atender al Cliente. Siguiendo en orden de importancia, están: Informar al cliente, Personalizar servicio, Estudiar patrón de comportamiento, y Gestionar catálogo.

6.1.2 Proceso de negocio seleccionado – Vender Producto/Servicio

Nosotros seleccionamos trabajar con el BP *Vender Producto/Servicio* (VPS), por su importancia para la estrategia CRM. La información que permite realizar el ejercicio de verificación sobre la colaboración Negocio a Negocio (*Business to Business*, B2B) de CRM que vamos a llevar a cabo se muestra en el BPD de Fig. 6.1.

El BP VPS define todas las actividades que permiten que una compañía realice la venta de un Producto/Servicio requerido por un cliente. En este sentido, el comprador o cliente (*Customer*) que requiere el Producto/Servicio, interactúa con la compañía (*Company*) para realizar el BP. De allí que podemos observar en la Fig. 6.1, siguiendo los lineamientos de BPMN [158], que el *Customer* esté representado por un *contenedor* y la *Company* por otro *contenedor*, los cuales intercambian *flujos de mensaje* para establecer la colaboración que requiere la realización del BP VPS. A su vez, el contenedor *Company* está particionado por los *Carriles* Ventas (*Sales*), Agente logístico (*Logistic agent*) y Canal de atención (*Attention channel*), representando los participantes internos de *Company* que intervienen en la realización del BP, de acuerdo a las siguientes responsabilidades:

- *Sales*. Coordina las actividades con el representante de la unidad comercial, ya que este implementa los lineamientos provenientes del mercadeo.
- *Attention channel*. Representa a la organización ante el Cliente; es con quien el cliente interactúa la mayoría de las veces.
- *Logistic agent*. Coordina, junto con la unidad de producción y cualquier unidad que lo requiera, las entregas, movilizaciones y requerimientos para cumplir las metas organizacionales.

El BP VPS se inicia cuando el *Customer* requiere una comunicación con *Company*. En este sentido, el BP VPS responde a necesidades del *Customer* para comprar algún Producto/Servicio. Sin embargo, si el BP VPS es iniciado por *Company* es porque responde a estrategias de CRM para entrar en contacto con los *Customer* para vender algún Producto/Servicio.

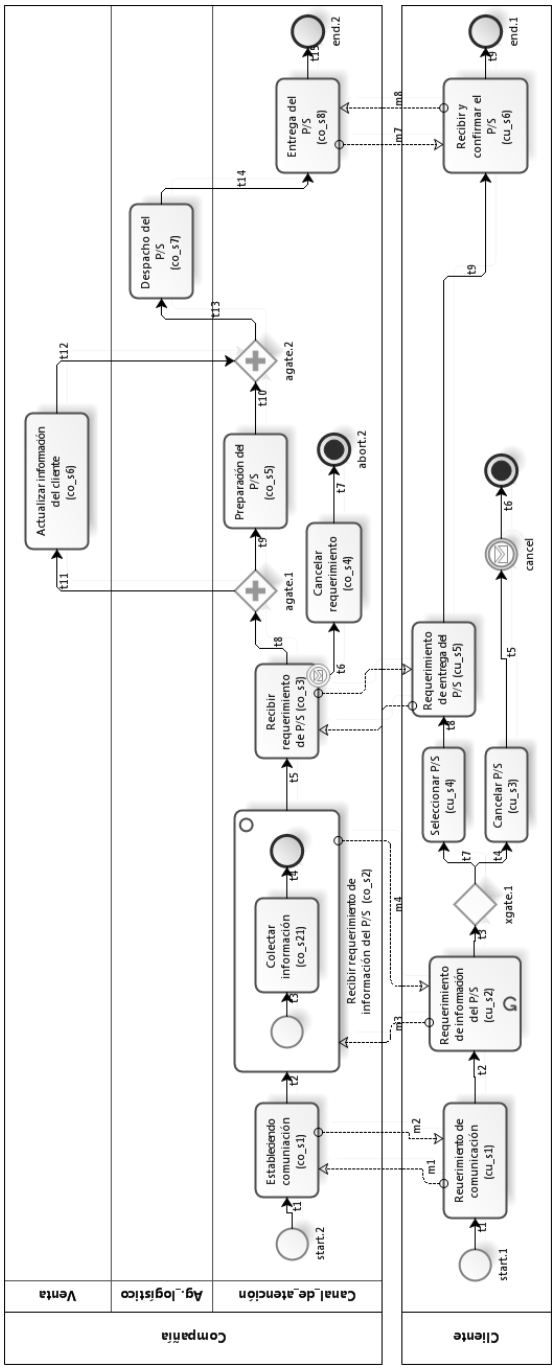


Fig. 6.1 BPD del BP Vender Producto/Servicio.

Como puede verse en la Fig. 6.1, este BP requiere una alta colaboración por parte de los participantes para lograr su realización, lo cual amerita una sincronización entre las actividades que forman parte de los flujos de mensaje.

A continuación veremos como nuestra propuesta permite: (1) verificar la correctitud del flujo de los procesos locales de cada participante (es decir, el *Customer* y la *Company*), de manera individual, aislados de cualquier colaboración, y (2) verificar la colaboración entre los participantes; es decir, comprobar la correctitud de la sincronización entre los procesos locales, la cual depende directamente del orden de ejecución de las actividades por parte de los participantes en el BP y de la sincronización de éstos a través de los flujos de mensaje.

6.1.3 Definición y descripción

Conocido el BP VPS (a través del BPD que lo define), se procede a obtener el BPTM; es decir, el modelo en CSP+T que muestra la sintaxis y la semántica del conjunto de elementos notacionales de BPMN que conforman los procesos locales *Customer* y *Company*, de acuerdo a la propuesta de reglas de transformación descritas en la sección 3.4.3. Dado que nuestra propuesta está más orientada a la semántica de los elementos notacionales de BPMN que representan tiempos de duración[1], la descripción sintáctica completa de los participantes *Customer* y *Company* están descritas en el Apéndice F y en esta sección nos centramos en la definición y descripción semántica del BPTM que especifica el comportamiento del BP VPS y que será más adelante objeto de verificación según nuestra propuesta.

Definimos los conjuntos *CU*, *CO* y *CO2*, para indexar los procesos del BPTM que corresponden a los elementos notaciones de los participantes *Customer*, *Company*, y del sub–proceso *co_s2*, respectivamente (ver Fig. 6.1).

$$CU = \{start.1, cu\_s1, cu\_s2, cu\_s3, cu\_s4, cu\_s5, cu\_s6, xgate.1, end.1, abort.1\}$$

[1] Aquí expresaremos los tiempos de duración en segundos, de acuerdo a la función*sec* (asumimos que un mes tiene 30 días y un año 365 días) [218]:

$$sec : Time \to \mathbb{N}$$

$$sec = \lambda\ Time \bullet$$
$$año * 31556926 + mes * 2629744 + día * 86400 + hora * 3600 + minuto * 60 + segundo$$

$$
\begin{aligned}
CO = \ & \{start.2, co\_s1, co\_s2, co\_s21, co\_s3, co\_s4, co\_s5, co\_s6, co\_s7, co\_s8, \\
& agate.1, agate.2, end.2, abort.2\} \\
CO2 = \ & \{start.3, co\_s21, end.3\}
\end{aligned}
$$

Los procesos *Cus*, *Com* y *SubCom* especificados en CSP+T (de acuerdo a la semántica presentada en la sección 3.4.3) son mostrado a continuación. El proceso *Cus* obtenido es como sigue:

$$
\begin{aligned}
Cus = \ & \textsf{let } X = \Box i : (\alpha Y \backslash \{fin.1, abt.1\}) \bullet \\
& \qquad (i \rightarrow X \,\Box\, fin.1 \rightarrow \textit{SKIP} \,\Box\, abt.1 \rightarrow \textit{STOP}) \\
& \quad Y = (\| i : CU \bullet \alpha P(i) \circ P(i)) \\
& \textsf{within } (Y \mid [\alpha Y] \mid X) \backslash \{\mid init.Cus \mid\}
\end{aligned}
$$

donde por cada $i \in CU$, los procesos $P(i)$ son definidos a continuación. Usamos $n \in \mathbb{N}$ para denotar el número de instancias de la actividad *Product/Service information request (cu_s2)*.

$$
\begin{aligned}
P(start.1) = \ & (t0.\star \rightarrow init.Cus.cu\_s1 \rightarrow \textit{SKIP}) \,\Box\, fin.1 \rightarrow \textit{SKIP} \\
P(cu\_s1) = \ & (init.Cus.cu\_s1 \rightarrow \textit{SKIP} \fatsemi starts.Cus.cu\_s1 \rightarrow \textit{SKIP} \fatsemi \\
& msg.cu\_s1!x : \{in, last\} \rightarrow \textit{SKIP} \fatsemi msg.cu\_s1.out \rightarrow \textit{SKIP} \fatsemi \\
& init.Cus.cu\_s2 \rightarrow \textit{SKIP} \fatsemi P(cu\_s1)) \,\Box\, fin.1 \rightarrow \textit{SKIP} \\
P(cu\_s2) = \ & \\
\textsf{let } A(n) = \ & n > 0 \ \& \ (init.Cus.cu\_s2 \rightarrow \textit{SKIP} \fatsemi starts.Cus.cu\_s2 \rightarrow \textit{SKIP} \fatsemi \\
& msg.cu\_s2!x : \{in, last\} \rightarrow \textit{SKIP} \fatsemi msg.cu\_s2.out \rightarrow \textit{SKIP} \fatsemi \\
& init.Cus.xgate.1 \rightarrow \textit{SKIP} \fatsemi A(n-1)) \,\Box\, init.Cus.xgate.1 \rightarrow \textit{SKIP} \\
X(n) = \ & (init.Cus.cu\_s2 \rightarrow \textit{SKIP} \,\Box\, init.Cus.xgate.1 \rightarrow \textit{SKIP}) \fatsemi \\
& (n > 1 \ \& \ (init.Cus.cu\_s2 \rightarrow (msg.cu\_s2.in \rightarrow X(n-1) \\
& \quad \Box\, msg.cu\_s2.last \rightarrow init.Cus.xgate.1 \rightarrow \textit{SKIP})) \\
& \Box \ n = 1 \ \& \ (init.Cus.cu\_s2 \rightarrow msg.cu\_s2.last \rightarrow \\
& \quad init.Cus.xgate.1 \rightarrow \textit{SKIP}) \\
& \Box \ n = N \ \& \ msg.cu\_s2.end \rightarrow init.Cus.xgate.1 \rightarrow \textit{SKIP}) \\
\textsf{within } & ((A(n) \mid [SynSet] \mid X(n)) \fatsemi P(cu\_s2)) \,\Box\, fin.1 \rightarrow \textit{SKIP} \\
& SynSet = \{msg.cu\_s2.in, msg.cu\_s2.last, init.Cus.cu\_s2, init.Cus.xgate.1\} \\
P(xgate.1) = \ & (init.Cus.xgate.1 \rightarrow \textit{SKIP} \fatsemi \\
& (init.Cus.cu\_s3 \rightarrow \textit{SKIP} \,\Box\, init.Cus.cu\_s4 \rightarrow \textit{SKIP}) \fatsemi \\
& P(xgate.1)) \,\Box\, fin.1 \rightarrow \textit{SKIP} \\
P(cu\_s3) = \ & (init.Cus.cu\_s3 \rightarrow \textit{SKIP} \fatsemi starts.Cus.cu\_s3 \rightarrow \textit{SKIP} \fatsemi \\
& init.Cus.cancel \rightarrow \textit{SKIP} \fatsemi P(cu\_s3)) \,\Box\, fin.1 \rightarrow \textit{SKIP} \\
P(cancel) = \ & (init.Cus.cancel \rightarrow \textit{SKIP} \fatsemi msg.cancel!x : \{can\} \rightarrow \textit{SKIP} \fatsemi \\
& msg.cancel.out \rightarrow \textit{SKIP} \fatsemi init.Cus.abort.1 \rightarrow \textit{SKIP} \fatsemi \\
& P(cancel)) \,\Box\, fin.1 \rightarrow \textit{SKIP} \\
P(abort.1) = \ & (init.Cus.abort.1 \rightarrow \textit{SKIP} \fatsemi abt.1 \rightarrow \textit{STOP}) \,\Box\, fin.1 \rightarrow \textit{SKIP} \\
P(cu\_s4) = \ & (init.Cus.cu\_s4 \rightarrow \textit{SKIP} \fatsemi starts.Cus.cu\_s4 \rightarrow \textit{SKIP} \fatsemi \\
& init.Cus.cu\_s5 \rightarrow \textit{SKIP} \fatsemi P(cu\_s4)) \,\Box\, fin.1 \rightarrow \textit{SKIP}
\end{aligned}
$$

$$
\begin{aligned}
P(cu\_s5) = {} & (init.Cus.cu\_s5 \rightarrow \mathit{SKIP} \fatsemi starts.Cus.cu\_s5 \rightarrow \mathit{SKIP} \fatsemi \\
& msg.cu\_s5!x : \{in, last\} \rightarrow \mathit{SKIP} \fatsemi msg.cu\_s5.out \rightarrow \mathit{SKIP} \fatsemi \\
& init.Cus.cu\_s6 \rightarrow \mathit{SKIP} \fatsemi P(cu\_s5)) \Box fin.1 \rightarrow \mathit{SKIP} \\
P(cu\_s6) = {} & (init.Cus.cu\_s6 \rightarrow \mathit{SKIP} \fatsemi starts.Cus.cu\_s6 \rightarrow \mathit{SKIP} \fatsemi \\
& msg.cu\_s6!x : \{in, last\} \rightarrow \mathit{SKIP} \fatsemi msg.cu\_s6.out \rightarrow \mathit{SKIP} \fatsemi \\
& init.Cus.end.1 \rightarrow \mathit{SKIP} \fatsemi P(cu\_s6)) \Box fin.1 \rightarrow \mathit{SKIP} \\
P(end.1) = {} & init.Cus.end.1 \rightarrow \mathit{SKIP} \fatsemi fin.1 \rightarrow \mathit{SKIP}
\end{aligned}
$$

El proceso *Com* obtenido es mostrado a continuación.

$$
\begin{aligned}
Com = {} & \mathsf{let}\ Z = \Box j : (\alpha R \backslash \{fin.2, abt.2\}) \bullet \\
& \qquad (j \rightarrow Z \Box fin.2 \rightarrow \mathit{SKIP} \Box abt.2 \rightarrow \mathit{STOP}) \\
R = {} & (\|j : CO \bullet \alpha P(j) \circ P(j)) \\
& \mathsf{within}\ (R \mid [\alpha R] \mid Z) \backslash \{\mid init.Com \mid\}
\end{aligned}
$$

donde para cada $j \in CO$, los procesos $P(j)$ son definidos a continuación.

$$
\begin{aligned}
P(start.2) = {} & (t0.\star \rightarrow init.Com.co\_s1 \rightarrow \mathit{SKIP}) \Box fin.2 \rightarrow \mathit{SKIP} \\
P(co\_s1) = {} & (init.Com.co\_s1 \bowtie vs1 \rightarrow \mathit{SKIP} \fatsemi starts.Com.co\_s1 \rightarrow \mathit{SKIP} \fatsemi \\
& msg.co\_s1!x : \{in, last\} \rightarrow \mathit{SKIP} \fatsemi msg.co\_s1.out \rightarrow \mathit{SKIP} \fatsemi \\
& \mathrm{I}(3600 - 1800, vs1 + 1800).init.Com.co\_s2 \rightarrow \mathit{SKIP} \fatsemi P(co\_s1)) \Box fin.2 \rightarrow \mathit{SKIP} \\
P(co\_s2) = {} & (init.Com.co\_s2 \bowtie vs2 \rightarrow \mathit{SKIP} \fatsemi msg.co\_s2!x : \{in, last\} \rightarrow \mathit{SKIP} \fatsemi \\
& msg.co\_s2.out \rightarrow \mathit{SKIP} \fatsemi starts.Com.co\_s2 \rightarrow \mathit{SKIP} \fatsemi \\
& (SubCom \mid [\{end.3\}] \mid end.3 \rightarrow \mathrm{I}(86400 - 64800, vs2 + 64800).init.Com.co\_s3 \rightarrow \mathit{SKIP}) \\
& \mid [\{init.Com.co\_s3\}] \mid \\
& \mathrm{I}(86400 - 64800, vs2 + 64800).init.Com.co\_s3 \rightarrow \mathit{SKIP}) \fatsemi P(co\_s2)) \\
& \Box fin.2 \rightarrow \mathit{SKIP} \\
P(co\_s3) = {} & (init.Com.co\_s3 \bowtie vs3 \rightarrow \mathit{SKIP} \fatsemi starts.Com.co\_s3 \rightarrow \\
& (\mathit{SKIP} \triangle (\mathrm{I}(600, vs3).msg.co\_s3?x : \{cancel\} \rightarrow \mathit{SKIP} \fatsemi init.Com.co\_s4 \rightarrow \mathit{SKIP}) \Box \\
& (msg.co\_s3!x : \{in, last\} \rightarrow \mathit{SKIP} \fatsemi msg.co\_s3.out \rightarrow \mathit{SKIP} \fatsemi \\
& \mathrm{I}(600, vs3).init.Com.agate.1 \rightarrow \mathit{SKIP}) \fatsemi P(co\_s3))) \Box fin.2 \rightarrow \mathit{SKIP} \\
P(co\_s4) = {} & (init.Com.co\_s4 \bowtie vs4 \rightarrow \mathit{SKIP} \fatsemi starts.Com.co\_s4 \rightarrow \mathit{SKIP} \fatsemi \\
& \mathrm{I}(600 - 300, vs4 + 300).init.Com.abort.2 \rightarrow \mathit{SKIP} \fatsemi P(co\_s4)) \Box fin.2 \rightarrow \mathit{SKIP} \\
P(abort.2) = {} & (init.Cus.abort.2 \rightarrow \mathit{SKIP} \fatsemi abt.2 \rightarrow \mathit{STOP}) \Box fin.2 \rightarrow \mathit{SKIP} \\
P(agate.1) = {} & (init.Com.agate.1 \rightarrow \mathit{SKIP} \fatsemi \\
& (init.Com.co\_s5 \rightarrow \mathit{SKIP} \| \| init.Com.co\_s6 \rightarrow \mathit{SKIP} \fatsemi \\
& P(agate.1)) \Box fin.2 \rightarrow \mathit{SKIP} \\
P(co\_s5) = {} & (init.Com.co\_s5 \bowtie vs5 \rightarrow \mathit{SKIP} \fatsemi starts.Com.co\_s5 \rightarrow \mathit{SKIP} \fatsemi \\
& \mathrm{I}(\min\{vs5 + (172800 - 86400), vs6 + 600\} - \\
& \max\{vs5 + 86400, vs6\}, \max\{vs5 + 86400, vs6\}).init.Com.agate.2 \rightarrow \mathit{SKIP} \fatsemi \\
& P(co\_s5)) \Box fin.2 \rightarrow \mathit{SKIP} \\
P(co\_s6) = {} & (init.Com.co\_s6 \bowtie vs6 \rightarrow \mathit{SKIP} \fatsemi starts.Com.co\_s6 \rightarrow \mathit{SKIP} \fatsemi \\
& \mathrm{I}(\min\{vs5 + (172800 - 86400), vs6 + 600\} - \\
& \max\{vs5 + 86400, vs6\}, \max\{vs5 + 86400, vs6\}).init.Com.agate.2 \rightarrow \mathit{SKIP} \fatsemi \\
& P(co\_s6)) \Box fin.2 \rightarrow \mathit{SKIP} \\
P(agate.2) = {} & ((P(co\_s5) \mid [\mathrm{I}(\min\{vs5 + (172800 - 86400), vs6 + 600\} - \\
& \max\{vs5 + 86400, vs6\}, \max\{vs5 + 86400, vs6\}).init.Com.agate.2] \mid P(co\_s6)) \rightarrow \mathit{SKIP} \fatsemi \\
& P(agate.2)) \Box fin.2 \rightarrow \mathit{SKIP}
\end{aligned}
$$

$$
\begin{aligned}
P(co\_s7) = {} & (init.Com.co\_s7 \bowtie vs7 \rightarrow \mathit{SKIP} \fatsemi starts.Com.co\_s7 \rightarrow \mathit{SKIP} \fatsemi \\
& \mathrm{I}(86400-64800, vs7+64800).init.Com.co\_s8 \rightarrow \mathit{SKIP} \fatsemi P(co\_s7)) \Box fin.2 \rightarrow \mathit{SKIP} \\
P(co\_s8) = {} & (init.Com.co\_s8 \bowtie vs8 \rightarrow \mathit{SKIP} \fatsemi starts.Com.co\_s8 \rightarrow \mathit{SKIP} \fatsemi \\
& msg.co\_s8!x : \{in, last\} \rightarrow \mathit{SKIP} \fatsemi msg.co\_s8.out \rightarrow \mathit{SKIP} \fatsemi \\
& \mathrm{I}(345600-86400, vs8+86400).init.Com.end.2 \rightarrow \mathit{SKIP} \fatsemi P(co\_s8)) \Box fin.2 \rightarrow \mathit{SKIP} \\
P(end.2) = {} & init.Com.end.2 \rightarrow \mathit{SKIP} \fatsemi fin.2 \rightarrow \mathit{SKIP}
\end{aligned}
$$

El proceso $SubCom$ describe la semántica del sub–proceso definido como parte de *co_s2* según su descripción sintáctica (ver Apéndice F).

$$
\begin{aligned}
SubCom = {} & \mathsf{let}\ T = \Box k : (\alpha W \backslash \{fin.3\}) \bullet (k \rightarrow T \Box fin.3 \rightarrow \mathit{SKIP}) \\
W = {} & (\|k : CO2 \bullet \alpha P(k) \circ P(k)) \\
& \mathsf{within}\ (W \mid [\alpha W] \mid T) \backslash \{|init|\}
\end{aligned}
$$

donde por cada $k \in CO2$, los procesos $P(k)$ son definidos a continuación.

$$
\begin{aligned}
P(start.3) = {} & (init.Com.co\_s21 \rightarrow \mathit{SKIP}) \Box fin.3 \rightarrow \mathit{SKIP} \\
P(co\_s21) = {} & (init.Com.co\_s21 \bowtie vs21 \rightarrow \mathit{SKIP} \fatsemi starts.Com.co\_s21 \rightarrow \mathit{SKIP} \fatsemi \\
& \mathrm{I}(1800, vs21).init.Com.end.3 \rightarrow \mathit{SKIP} \fatsemi P(co\_s21)) \Box fin.3 \rightarrow \mathit{SKIP} \\
P(end.3) = {} & init.Com.end.3 \rightarrow \mathit{SKIP} \fatsemi fin.3 \rightarrow \mathit{SKIP}
\end{aligned}
$$

La colaboración entre los participantes *Customer* y *Company* es la composición paralela de los procesos *Cus* y *Com*, tal como se denota a continuación por el término de proceso CSP+T PVPS:

$$PVPS = (Cus \mid [\alpha Cus \| \alpha Com] \mid Com) \backslash \{|msg|\}$$

El conjunto de términos de procesos descritos anteriormente (Cus, Com y $PVPS$), conforman el BPTM o modelo en CSP+T del BP VPS. En este sentido es elmodelo que verificaremos respecto de las propiedades especificadas en CCTL, que se espera cumpla este modelo y que se presentan en la siguiente sección.

6.1.4 Comportamiento esperado – definición de propiedades

En en el Apéndice F están compiladas algunas propiedades especificadas en CCTL que debe cumplir el BP VPS y que han sido derivadas de reglas de negocio relacionadas con la estrategia CRM y que están de acuerdo con la descripción general de CRM hecha al comienzo de la sección 6.1. Para mostrar la aplicación de nuestra propuesta, trabajaremos con la propiedad ϕ_3 (ver Apéndice F), la cual está relacionada con la *obligación de obtener la*

recepción y confirmación de entrega del Producto/Servicio, una vez que el Customer ha iniciado la comunicación con Company.

Como haremos la verificación del comportamiento del BPTM derivado del BP VPS (denotado anteriomente como $PVPS$) a partir de los sub–procesos que lo conforman (es decir, Cus y Com), aplicando nuestro enfoque de verificación composicional, entonces debemos definir las propiedades que debe cumplir cada participante, las cuales muestran la secuencia de eventos esperada cuando ellos ejecutan los procesos parciales que están bajo su responsabilidad. Los participantes deben ejecutar todas sus actividades en el orden esperado para lograr que el proceso global funcione. En la Tabla 6.1 están definidas las propiedades parciales que debemos verificar en los procesos Cus y Com, respectivamente, para obtener la verificación del proceso $PVPS$. Los TBAs mostrados en la Tabla 6.1 fueron obtenidos como resultado de la aplicación del algoritmo descrito en la sección 3.5.1.

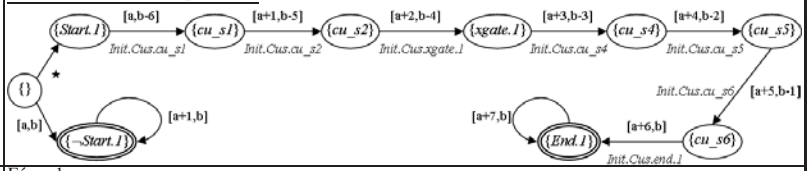
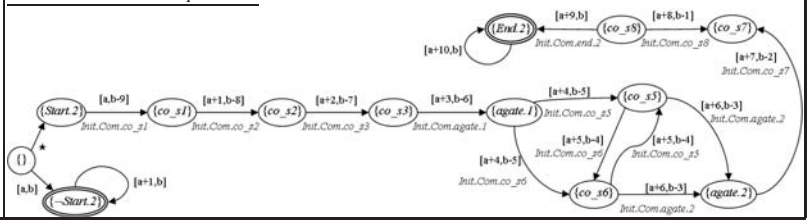
Propiedad	Especificación
ϕ_{Cus}	Fórmula: $\mathsf{AG}_{[a,b]}(Start.1 \rightarrow \mathsf{A}[cu\_s1 \ \mathsf{U}_{[a+1,b-5]} \ (cu\_s2 \wedge \mathsf{A}[cu\_s2 \ \mathsf{U}_{[a+2,b-4]} \ (xgate.1 \wedge \mathsf{A}[xgate.1 \ \mathsf{U}_{[a+3,b-3]} \ (cu\_s4 \wedge \mathsf{A}[cu\_s4 \ \mathsf{U}_{[a+4,b-2]} \ (cu\_s5 \wedge \mathsf{A}[cu\_s5 \ \mathsf{U}_{[a+5,b-1]} \ (cu\_s6 \wedge \mathsf{A}[cu\_s6 \ \mathsf{U}_{[a+6,b]} \ End.1])])])])])])$ TBA semánticamente equivalente: 
ϕ_{Com}	Fórmula: $\mathsf{AG}_{[a,b]}(Start.2 \rightarrow \mathsf{A}[co\_s1 \ \mathsf{U}_{[a+1,b-8]} \ (co\_s2 \wedge \mathsf{A}[cu\_s2 \ \mathsf{U}_{[a+2,b-7]} \ (co\_s3 \wedge \mathsf{A}[co\_s3 \ \mathsf{U}_{[a+3,b-6]} \ (agate.1 \wedge \mathsf{A}[agate.1 \ \mathsf{U}_{[a+4,b-5]} \ (\{co\_s5 \vee co\_s6\} \wedge \mathsf{A}[\{co\_s5 \vee co\_s6\} \ \mathsf{U}_{[a+6,b-3]} \ (agate.2 \wedge \mathsf{A}[agate.2 \ \mathsf{U}_{[a+7,b-2]} \ (co\_s7 \wedge \mathsf{A}[co\_s7 \ \mathsf{U}_{[a+8,b-1]} \ (co\_s8 \wedge \mathsf{A}[co\_s8 \ \mathsf{U}_{[a+9,b]} \ End.2])])])])])])])])$ TBA semánticamente equivalente: 

Tabla 6.1 Propiedad de los participantes *Customer* (*Cus*) y *Company* (*Com*).

La Tabla 6.1 muestra la interpretación a través de TBAs de la propiedad ϕ_3 (ver Apéndice F), para cada participante. Así, cuando el BP VPS se ejecuta, el proceso Cus que representa el participante *Customer*, pasa por los siguientes estados: $Start.1$, cu_s1, cu_s2, $xgate.1$, cu_s4, cu_s5, cu_s6 y $End.1$; mientras que el proceso Com que representa al particpante *Com-*

pany, pasa por los estados: $Start.2$, co_s1, co_s2, co_s3, $agate.1$, co_s5, co_s6, $agate.2$, co_s7, co_s8, $End.2$ (ver Tabla 6.1).

Usando las reglas de transformación indicadas en la Definición 3.8 (sección 3.5.2), obtenemos la interpretación operacional de los TBA semánticamente equivalentes a las fórmulas CCTL previamente especificadas, de acuerdo al cálculo de procesos CSP+T, mostrados en la Tabla 6.2. Estos procesos describen el comportamiento esperado para los participantes que colaboran en el BP VPS en el mismo dominio semántico del BPTM.

TBA	Término de proceso CSP+T
ϕ_{Cus}	$T(\phi_{Cus}) = \ t_0.\star \rightarrow T(Start.1)$ $T(Start.1) = \ \mathrm{I}((b-6)-a, a).init.Cus.cu\_s1 \rightarrow T(cu\_s1)$ $T(cu\_s1) = \ \mathrm{I}((b-5)-(a+1), a+1).init.Cus.cu\_s2 \rightarrow T(cu\_s2)$ $T(cu\_s2) = \ \mathrm{I}((b-4)-(a+2), a+2).init.Cus.xgate.1 \rightarrow T(xgate.1))$ $T(xgate.1) = \mathrm{I}((b-3)-(a+3), a+3).init.Cus.cu\_s4 \rightarrow T(cu\_s4)$ $T(cu\_s4) = \ \mathrm{I}((b-2)-(a+4), a+4).init.Cus.cu\_s5 \rightarrow T(cu\_s5)$ $T(cu\_s5) = \ \mathrm{I}((b-1)-(a+5), a+5).init.Cus.cu\_s6 \rightarrow T(cu\_s6)$ $T(cu\_s6) = \ \mathrm{I}(b-(a+6), a+6).init.Cus.end.1 \rightarrow T(End.1)$ $T(End.1) = \ SKIP \fatsemi T(\phi_{Cus})$
ϕ_{Com}	$T(\phi_{Com}) = \ t_0.\star \rightarrow T(Start.2)$ $T(Start.2) = \ \mathrm{I}((b-9)-a, a).init.Com.co\_s1 \rightarrow T(co\_s1)$ $T(co\_s1) = \ \mathrm{I}((b-8)-(a+1), a+1).init.Com.co\_s2 \rightarrow T(co\_s2)$ $T(co\_s2) = \ \mathrm{I}((b-7)-(a+2), a+2).init.Com.co\_s3 \rightarrow T(co\_s3))$ $T(co\_s3) = \ \mathrm{I}((b-6)-(a+3), a+3).init.Com.agate.1 \rightarrow T(agate.1)$ $T(agate.1) = (\mathrm{I}((b-5)-(a+4), a+4).init.Com.co\_s5 \rightarrow T(co\_s5)) \ \Box$ $(\mathrm{I}((b-5)-(a+4), a+4).init.Com.co\_s6 \rightarrow T(co\_s6))$ $T(co\_s5) = \ (\mathrm{I}((b-4)-(a+5), a+5).init.Com.co\_s6 \rightarrow T(co\_s6)) \ \Box$ $(\mathrm{I}((b-3)-(a+6), a+6).init.Com.agate.2 \rightarrow T(agate.2))$ $T(cu\_s6) = \ (\mathrm{I}((b-4)-(a+5), a+5).init.Com.co\_s5 \rightarrow T(co\_s5)) \ \Box$ $(\mathrm{I}((b-3)(a+6), a+6).init.Com.agate.2 \rightarrow T(agate.2))$ $T(agate.2) = \mathrm{I}((b-2)-(a+7), a+7).init.Com.co\_s7 \rightarrow T(co\_s7)$ $T(co\_s7) = \ \mathrm{I}((b-1)-(a+8), a+8).init.Com.co\_s8 \rightarrow T(co\_s8)$ $T(co\_s8) = \ \mathrm{I}((b)-(a+9), a+9).init.Com.end.2 \rightarrow T(End.2)$ $T(End.2) = \ SKIP \fatsemi T(\phi_{Com})$

Tabla 6.2 Términos de proceso CSP+T que corresponden a los TBA de las propiedades.

6.1.5 Verificando la colaboración

Una vez que se obtienen el BPTM y las propiedades que éste debe cumplir, en el mismo dominio semántico, se puede proceder a la verificación del BPTM derivado del BP VPS. Según nuestro enfoque, debemos verificar que los procesos que representan el comportamiento de los participantes en el BPTM (es decir, Cus y Com) cumplen con las propiedades especificadas en la sección 6.1.4. Luego, bajo el dominio semántico del cálculo basado en CSP, se comprueba que se cumplen las siguientes relaciones de refinamiento:

$$T(\phi_{Cus}) \sqsubseteq_T Cus \ , \ T(\phi_{Com}) \sqsubseteq_T Com \tag{6.1}$$

$$T(\phi_{Cus}) \sqsubseteq_F Cus \ , \ T(\phi_{Com}) \sqsubseteq_F Com \tag{6.2}$$

Para verificar las relaciones anteriores, vamos a trabajar bajo el modelo semántico no temporizado de CSP, ya que, como indicamos en la sección 2.2.3.3, las propiedades de seguridad y vivacidad no temporizadas de un sistema temporizado deberán ser verificables en el modelo no temporizado y luego ser usadas en el análisis temporizado. Además, esto nos permite integrar el uso de la herramienta FDR2 para llevar a cabo la verificación de los procesos que representan a los participantes. En el Apéndice F se especifican los términos de proceso CSP $UT(Com)$, $UT(Cus)$, $UT(\phi_{Com})$, y $UT(\phi_{Cus})$, que corresponden a los procesos no temporizados de los participantes *Customer* y *Company*, y de las propiedades que éstos deben cumplir, respectivamente.

Vamos a utilizar las Definiciones 2.14 y 2.15 (ver sección 2.2.3.3) para comprobar que:

$$UT(\phi_{Cus}) \sqsubseteq_t T(\phi_{Cus}) \ , \ UT(\phi_{Com}) \sqsubseteq_t T(\phi_{Com}) \tag{6.3}$$
$$UT(Cus) \sqsubseteq_t Cus \ , \ UT(Com) \sqsubseteq_t Com \tag{6.4}$$
$$UT(\phi_{Cus}) \sqsubseteq_f T(\phi_{Cus}) \ , \ UT(\phi_{Com}) \sqsubseteq_f T(\phi_{Com}) \tag{6.5}$$
$$UT(Cus) \sqsubseteq_f Cus \ , \ UT(Com) \sqsubseteq_f Com \tag{6.6}$$

Una manera sencilla de demostrar las relaciones (6.1) a (6.6), es por medio del *desdoblado* (*unfolding*) de la definición de cada proceso sobre la base de las reglas algebraicas asociadas con los operadores de CSP [92] y CSP+T [209]. Este enfoque conduce a la exploración manual o la enumeración de todos los posibles estados de la definición de un proceso. Además, dado que nuestro interés está en las propiedades comportamentales, podemos trabajar en una versión abstracta de cada proceso que es independiente del paso de mensajes a través de los canales de comunicación [224].

El desarrollo de los términos Cus, Com, $UT(Cus)$ y $UT(Com)$, respectivamente, corresponden a las expresiones que se indican a continuación. Téngase en cuenta que los eventos $end.i$, $fin.i$ y $abt.i$, utilizados en la descripción semántica de los procesos, sólo son utilizados para el control interno de la ejecución de los procesos como parte del modelo operacional de CSP y no forman parte del alfabeto de comunicación que corresponde a cada uno de los participantes. Adicionalmente, los eventos $starts.Cus.x$ y $starts.Com.y$, son utilizados para representar *la ejecución de las actividades*, por lo que tampoco forman parte del alfabeto de comunicación de los procesos que representan a los participantes. Por lo indicado anteriormente,

sólo colocamos en las siguientes expresiones los eventos que forman parte de la interfaz de cada proceso.

$$
\begin{aligned}
Cus \;=\; & t_0.\star \to init.Cus.cu\_s1 \to init.Cus.cu\_s2 \to init.Cus.xgate.1 \to \\
& \left(\begin{array}{l} init.Cus.cu\_s3 \to init.Cus.cancel \to init.Cus.abort.1 \to STOP \\ \Box \\ init.Cus.cu\_s4 \to init.Cus.cu\_s5 \to init.Cus.cu\_s6 \to init.Cus.end.1 \to Cus \end{array} \right)
\end{aligned}
$$

$$
\begin{aligned}
Com \;=\; & t_0.\star \to init.Com.co\_s1 \to \mathrm{I}(3600-1800, vs1+1800).init.Com.co\_s2 \to \\
& \mathrm{I}(86400-64800, vs2+64800).init.Com.co\_s3 \to \\
& \left(\begin{array}{l} init.Com.co\_s4 \to init.Com.cancel \to \mathrm{I}(600-300, vs4+300).init.Com.abort.2 \\ \to STOP \\ \Box \\ \mathrm{I}(600, vs3).init.Com.agate.1 \to \left(\begin{array}{l} init.Com.co\_s5 \to init.Com.co\_s6 \\ \Box \\ init.Com.co\_s6 \to init.Com.co\_s5 \end{array} \right) \to \\ \mathrm{I}(\min\{vs5+(172800-86400), vs6+600\} - \max\{vs5+86400, vs6\}, \\ \quad \max\{vs5+86400, vs6\}).init.Com.agate.2 \to \\ init.Com.co\_s7 \to \mathrm{I}(86400-64800, vs7+64800).init.Com.co\_s8 \to \\ \mathrm{I}(345600-86400, vs8+86400).init.Com.end.2 \to Com \end{array} \right)
\end{aligned}
$$

$$
\begin{aligned}
UT(Cus) \;=\; & \star \to init.Cus.cu\_s1 \to init.Cus.cu\_s2 \to init.Cus.xgate.1 \to \\
& \left(\begin{array}{l} init.Cus.cu\_s3 \to init.Cus.cancel \to init.Cus.abort.1 \to STOP \\ \Box \\ init.Cus.cu\_s4 \to init.Cus.cu\_s5 \to init.Cus.cu\_s6 \to init.Cus.end.1 \\ \to UT(Cus) \end{array} \right)
\end{aligned}
$$

$$
\begin{aligned}
UT(Com) \;=\; & \star \to init.Com.co\_s1 \to init.Com.co\_s2 \to init.Com.co\_s3 \to \\
& \left(\begin{array}{l} init.Com.co\_s4 \to init.Com.cancel \to init.Com.abort.2 \to STOP \\ \Box \\ init.Com.agate.1 \to \left(\begin{array}{l} init.Com.co\_s5 \to init.Com.co\_s6 \\ \Box \\ init.Com.co\_s6 \to init.Com.co\_s5 \end{array} \right) \to \\ init.Com.agate.2 \to init.Com.co\_s7 \to init.Com.co\_s8 \to \\ init.Com.end.2 \to UT(Com) \end{array} \right)
\end{aligned}
$$

El desarrollo de los términos $T(\phi_{Cus})$, $T(\phi_{Com})$, $UT(\phi_{Cus})$ y $UT(\phi_{Com})$ son los siguientes:

$$
\begin{aligned}
T(\phi_{Cus}) \;=\; & t_0.\star \to \mathrm{I}((b-6)-a, a).init.Cus.cu\_s1 \to \mathrm{I}((b-5)-(a+1), a+1).init.Cus.cu\_s2 \to \\
& \mathrm{I}((b-4)-(a+2), a+2).init.Cus.xgate.1 \to \mathrm{I}((b-3)-(a+3), a+3).init.Cus.cu\_s4 \to \\
& \mathrm{I}((b-2)-(a+4), a+4).init.Cus.cu\_s5 \to \mathrm{I}((b-1)-(a+5), a+5).init.Cus.cu\_s6 \to \\
& \mathrm{I}(b-(a+6), a+6).init.Cus.end.1 \to T(\phi_{Cus})
\end{aligned}
$$

$$
\begin{aligned}
T(\phi_{Com}) \;=\; & t_0.\star \to \mathrm{I}((b-9)-a, a).init.Com.co\_s1 \to \mathrm{I}((b-8)-(a+1), a+1).init.Com.co\_s2 \to \\
& \mathrm{I}((b-7)-(a+2), a+2).init.Com.co\_s3 \to \mathrm{I}((b-6)-(a+3), a+3).init.Com.agate.1 \to \\
& \left(\begin{array}{l} \mathrm{I}((b-5)-(a+4), a+4).init.Com.co\_s5 \to \mathrm{I}((b-4)-(a+5), a+5).init.Com.co\_s6 \\ \Box \\ \mathrm{I}((b-5)-(a+4), a+4).init.Com.co\_s6 \to \mathrm{I}((b-4)-(a+5), a+5).init.Com.co\_s5 \end{array} \right) \to \\
& \mathrm{I}((b-3)(a+6), a+6).init.Com.agate.2 \to \mathrm{I}((b-2)-(a+7), a+7).init.Com.co\_s7 \to \\
& \mathrm{I}((b-1)-(a+8), a+8).init.Com.co\_s8 \to \mathrm{I}((b)-(a+9), a+9).init.Com.end.2 \to T(\phi_{Com})
\end{aligned}
$$

$$UT(\phi_{Cus}) = \star \to init.Cus.cu\_s1 \to init.Cus.cu\_s2 \to init.Cus.xgate.1 \to init.Cus.cu\_s4 \to init.Cus.cu\_s5 \to init.Cus.cu\_s6 \to init.Cus.end.1 \to T(\phi_{Cus})$$

$$UT(\phi_{Com}) = \star \to init.Com.co\_s1 \to init.Com.co\_s2 \to init.Com.co\_s3 \to init.Com.agate.1 \to \left(\begin{array}{l} init.Com.co\_s5 \to init.Com.co\_s6 \\ \Box \\ init.Com.co\_s6 \to init.Com.co\_s5 \end{array}\right) \to init.Com.agate.2 \to init.Com.co\_s7 \to init.Com.co\_s8 \to init.Com.end.2 \to T(\phi_{Com})$$

Aplicando la Definición 2.14, obtenemos la verificación de las relaciones (6.3) y (6.4), según el modelo de trazas no temporizadas de CSP. Sea,

$$strip(T(\phi_{Cus})) = \langle \star, init.Cus.cu\_s1, init.Cus.cu\_s2, init.Cus.xgate.1, init.Cus.cu\_s4, init.Cus.cu\_s5, init.Cus.cu\_s6, init.Cus.end.1 \rangle$$

$$strip(T(\phi_{Com})) = \langle \star, init.Com.co\_s1, init.Com.co\_s2, init.Com.co\_s3, init.Com.agate.1, init.Com.co\_s5, init.Com.co\_s6, init.Com.agate.2, init.Com.co\_s7, init.Com.co\_s8, init.Com.end.2 \rangle, \langle \star, init.Com.co\_s1, init.Com.co\_s2, init.Com.co\_s3, init.Com.agate.1, init.Com.co\_s6, init.Com.co\_s5, init.Com.agate.2, init.Com.co\_s7, init.Com.co\_s8, init.Com.end.2 \rangle$$

$$strip(Cus) = \langle \star, init.Cus.cu\_s1, init.Cus.cu\_s2, init.Cus.xgate.1, init.Cus.cu\_s3, init.Cus.cancel, init.Cus.abort.1 \rangle, \langle \star, init.Cus.cu\_s1, init.Cus.cu\_s2, init.Cus.xgate.1, init.Cus.cu\_s4, init.Cus.cu\_s5, init.Cus.cu\_s6, init.Cus.end.1 \rangle$$

$$strip(Com) = \langle \star, init.Com.co\_s1, init.Com.co\_s2, init.Com.co\_s3, init.Com.co\_s4, init.Com.cancel, init.Com.abort.2 \rangle, \langle \star, init.Com.co\_s1, init.Com.co\_s2, init.Com.co\_s3, init.Com.agate.1, init.Com.co\_s5, init.Com.co\_s6, init.Com.agate.2, init.Com.co\_s7, init.Com.co\_s8, init.Com.end.2 \rangle, \langle \star, init.Com.co\_s1, init.Com.co\_s2, init.Com.co\_s3, init.Com.agate.1, init.Com.co\_s6, init.Com.co\_s5, init.Com.agate.2, init.Com.co\_s7, init.Com.co\_s8, init.Com.end.2 \rangle$$

$$\mathsf{traces}(UT(\phi_{Cus})) = \{\langle\rangle, \langle \star, init.Cus.cu\_s1, init.Cus.cu\_s2, init.Cus.xgate.1, init.Cus.cu\_s4, init.Cus.cu\_s5, init.Cus.cu\_s6, init.Cus.end.1 \rangle\} \quad (6.7)$$

$$\mathsf{traces}(UT(\phi_{Com})) = \{\langle\rangle, \langle \star, init.Com.co\_s1, init.Com.co\_s2, init.Com.co\_s3, init.Com.agate.1, init.Com.co\_s5, init.Com.co\_s6, init.Com.agate.2, init.Com.co\_s7, init.Com.co\_s8, init.Com.end.2 \rangle, \langle \star, init.Com.co\_s1, init.Com.co\_s2, init.Com.co\_s3, init.Com.agate.1, init.Com.co\_s6, init.Com.co\_s5, init.Com.agate.2, init.Com.co\_s7, init.Com.co\_s8, init.Com.end.2 \rangle\} \quad (6.8)$$

$$\mathsf{traces}(UT(Cus)) = \{\langle\rangle, \langle \star, init.Cus.cu\_s1, init.Cus.cu\_s2, init.Cus.xgate.1, init.Cus.cu\_s3, init.Cus.cancel, init.Cus.abort.1 \rangle, \langle \star, init.Cus.cu\_s1, init.Cus.cu\_s2, init.Cus.xgate.1, init.Cus.cu\_s4, init.Cus.cu\_s5, init.Cus.cu\_s6, init.Cus.end.1 \rangle\} \quad (6.9)$$

$$\mathsf{traces}(UT(Com)) = \{\langle\rangle, \langle \star, init.Com.co\_s1, init.Com.co\_s2, init.Com.co\_s3, init.Com.co\_s4, init.Com.cancel, init.Com.abort.2 \rangle, \langle \star, init.Com.co\_s1, init.Com.co\_s2, init.Com.co\_s3, init.Com.agate.1, init.Com.co\_s5, init.Com.co\_s6, init.Com.agate.2, init.Com.co\_s7, init.Com.co\_s8, init.Com.end.2 \rangle, \langle \star, init.Com.co\_s1, init.Com.co\_s2, init.Com.co\_s3, init.Com.agate.1, init.Com.co\_s6, init.Com.co\_s5, init.Com.agate.2, init.Com.co\_s7, \quad (6.10)$$

$$init.Com.co\_s8, init.Com.end.2\rangle\}\ ,$$

escribimos:

$$\begin{aligned}
strip(\,T(\phi_{Cus})) \in \mathsf{traces}(\,UT(\phi_{Cus})) &\Leftrightarrow\ UT(\phi_{Cus}) \sqsubseteq_t\ T(\phi_{Cus}),\\
strip(\,T(\phi_{Com})) \in \mathsf{traces}(\,UT(\phi_{Com})) &\Leftrightarrow\ UT(\phi_{Com}) \sqsubseteq_t\ T(\phi_{Com}),\\
strip(Cus) \in \mathsf{traces}(\,UT(Cus)) &\Leftrightarrow\ UT(Cus) \sqsubseteq_t\ Cus,\\
strip(Com) \in \mathsf{traces}(\,UT(Com)) &\Leftrightarrow\ UT(Com) \sqsubseteq_t\ Com,
\end{aligned}$$

que corresponden a las relaciones (6.3) y (6.4). Esto significa que los términos de proceso CSP+T, tanto del BPTM como de las propiedades esperadas, son un refinamiento hacia trazas temporizadas de los términos de proceso CSP propuestos.

Aplicando la Definición 2.15, podemos realizar la verificación de las relaciones (6.5) y (6.6), según el modelo de fallos no temporizado de CSP. Sea,

$$\begin{aligned}
\mathbf{SF}[\![\,T(\phi_{Cus})]\!] &= \{(\langle\star\rangle, \{init.Cus.cu_s1\}), (\langle\star, init.Cus.cu_s1, init.Cus.cu_s2, init.Cus.xgate.1,\\
&\quad init.Cus.cu_s4, init.Cus.cu_s5, init.Cus.cu_s6, init.Cus.end.1\rangle, \{\})\}\\
\mathbf{SF}[\![\,T(\phi_{Com})]\!] &= \{(\langle\star\rangle, \{init.Com.co_s1\}), (\langle\star, init.Com.co_s1, init.Com.co_s2, init.Com.co_s3,\\
&\quad init.Com.agate.1, init.Com.co_s5, init.Com.co_s6, init.Com.agate.2,\\
&\quad init.Com.co_s7, init.Com.co_s8, init.Com.end.2\rangle, \{\}), (\langle\star, init.Com.co_s1,\\
&\quad init.Com.co_s2, init.Com.co_s3, init.Com.agate.1, init.Com.co_s6,\\
&\quad init.Com.co_s5, init.Com.agate.2, init.Com.co_s7, init.Com.co_s8,\\
&\quad init.Com.end.2\rangle, \{\})\}\\
\mathbf{SF}[\![\,Cus]\!] &= \{(\langle\star\rangle, \{init.Cus.cu_s1\}), (\langle\star, init.Cus.cu_s1, init.Cus.cu_s2, init.Cus.xgate.1,\\
&\quad init.Cus.cu_s3, init.Cus.cancel, init.Cus.abort.1\rangle, \{\}), (\langle\star, init.Cus.cu_s1,\\
&\quad init.Cus.cu_s2, init.Cus.xgate.1, init.Cus.cu_s4, init.Cus.cu_s5,\\
&\quad init.Cus.cu_s6, init.Cus.end.1\rangle, \{\})\}\\
\mathbf{SF}[\![\,Com]\!] &= \{(\langle\star\rangle, \{init.Com.co_s1\}), (\langle\star, init.Com.co_s1, init.Com.co_s2, init.Com.co_s3,\\
&\quad init.Com.co_s4, init.Com.cancel, init.Com.abort.2\rangle, \{\}), (\langle\star, init.Com.co_s1,\\
&\quad init.Com.co_s2, init.Com.co_s3, init.Com.agate.1, init.Com.co_s5,\\
&\quad init.Com.co_s6, init.Com.agate.2, init.Com.co_s7, init.Com.co_s8,\\
&\quad init.Com.end.2\rangle, \{\}), (\langle\star, init.Com.co_s1, init.Com.co_s2, init.Com.co_s3,\\
&\quad init.Com.agate.1, init.Com.co_s6, init.Com.co_s5, init.Com.agate.2,\\
&\quad init.Com.co_s7, init.Com.co_s8, init.Com.end.2\rangle, \{\})\}\\
\mathbf{SF}[\![\,UT(\phi_{Cus})]\!] &= \{(\langle\star\rangle, \{init.Cus.cu_s1\}), (\langle\star, init.Cus.cu_s1, init.Cus.cu_s2, init.Cus.xgate.1,\\
&\quad init.Cus.cu_s4, init.Cus.cu_s5, init.Cus.cu_s6, init.Cus.end.1\rangle, \{\})\} \qquad (6.11)\\
\mathbf{SF}[\![\,UT(\phi_{Com})]\!] &= \{(\langle\star\rangle, \{init.Com.co_s1\}), (\langle\star, init.Com.co_s1, init.Com.co_s2, init.Com.co_s3,\\
&\quad init.Com.agate.1, init.Com.co_s5, init.Com.co_s6, init.Com.agate.2,\\
&\quad init.Com.co_s7, init.Com.co_s8, init.Com.end.2\rangle, \{\}), (\langle\star, init.Com.co_s1,\\
&\quad init.Com.co_s2, init.Com.co_s3, init.Com.agate.1, init.Com.co_s6,\\
&\quad init.Com.co_s5, init.Com.agate.2, init.Com.co_s7, init.Com.co_s8,\\
&\quad init.Com.end.2\rangle, \{\})\} \qquad (6.12)\\
\mathbf{SF}[\![\,UT(Cus)]\!] &= \{(\langle\star\rangle, \{init.Cus.cu_s1\}), (\langle\star, init.Cus.cu_s1, init.Cus.cu_s2, init.Cus.xgate.1,\\
&\quad init.Cus.cu_s4, init.Cus.cu_s5, init.Cus.cu_s6, init.Cus.end.1\rangle, \{\})\} \qquad (6.13)
\end{aligned}$$

$$
\begin{aligned}
& init.Cus.cu\_s3, init.Cus.cancel, init.Cus.abort.1\rangle, \{\}), \langle \star, init.Cus.cu\_s1, \\
& init.Cus.cu\_s2, init.Cus.xgate.1, init.Cus.cu\_s4, init.Cus.cu\_s5, \\
& init.Cus.cu\_s6, init.Cus.end.1\rangle, \{\})\} \\
\mathbf{SF}[\![UT(Com)]\!] = \{ & (\langle \star \rangle, \{init.Com.co\_s1\}), (\langle \star, init.Com.co\_s1, init.Com.co\_s2, init.Com.co\_s3, \\
& init.Com.co\_s4, init.Com.cancel, init.Com.abort.2\rangle, \{\}), (\langle \star, init.Com.co\_s1, \\
& init.Com.co\_s2, init.Com.co\_s3, init.Com.agate.1, init.Com.co\_s5, \\
& init.Com.co\_s6, init.Com.agate.2, init.Com.co\_s7, init.Com.co\_s8, \\
& init.Com.end.2\rangle, \{\}), \langle \star, init.Com.co\_s1, init.Com.co\_s2, init.Com.co\_s3, \\
& iinit.Com.agate.1, nit.Com.co\_s6, init.Com.co\_s5, init.Com.agate.2, \\
& init.Com.co\_s7, init.Com.co\_s8, init.Com.end.2\rangle, \{\})\}
\end{aligned} \tag{6.14}
$$

escribimos:

$$
\begin{aligned}
\mathbf{SF}[\![T(\phi_{Cus})]\!] \in \mathbf{SF}[\![UT(\phi_{Cus})]\!] &\Leftrightarrow UT(\phi_{Cus}) \sqsubseteq_f T(\phi_{Cus}) \\
\mathbf{SF}[\![T(\phi_{Com})]\!] \in \mathbf{SF}[\![UT(\phi_{Com})]\!] &\Leftrightarrow UT(\phi_{Com}) \sqsubseteq_f T(\phi_{Com}), \\
\mathbf{SF}[\![Cus]\!] \in \mathbf{SF}[\![UT(Cus)]\!] &\Leftrightarrow UT(Cus) \sqsubseteq_f Cus \\
\mathbf{SF}[\![Com]\!] \in \mathbf{SF}[\![UT(Com)]\!] &\Leftrightarrow UT(Com) \sqsubseteq_f Com,
\end{aligned}
$$

que corresponden a las relaciones (6.5) y (6.6). Esto significa que los términos de proceso CSP+T, tanto de los procesos que representan a los participantes en el BPTM como de las propiedades esperadas, son un refinamiento hacia fallos temporizados de los términos de proceso CSP propuestos. Además, por las relaciones (6.8) a (6.11), escribimos:

$$
\mathsf{traces}(UT(Cus)) \in \mathsf{traces}(UT(\phi_{Cus})) \Leftrightarrow UT(\phi_{Cus}) \sqsubseteq_T UT(Cus), \tag{6.15}
$$

$$
\mathsf{traces}(UT(Com)) \in \mathsf{traces}(UT(\phi_{Com})) \Leftrightarrow UT(\phi_{Com}) \sqsubseteq_T UT(Com), \tag{6.16}
$$

lo que corresponde a la verificación del comportamiento especificado de Cus y Com en términos de CSP (es decir, $UT(Cus)$ y $UT(Com)$) de acuerdo con el comportamiento esperado para este participante, también especificada en CSP (es decir, $UT(\phi_{Cus})$ y $UT(\phi_{Com})$). Como resultado, el comportamiento de los procesos Cus y Com que representan a los participantes *Customer* y *Company*, respectivamente, especificados en CSP fueron verificados con respeto al campo semántico de trazas, lo que asegura que se cumplen las propiedades de seguridad.

También, por las relaciones (6.12) a (6.15), escribimos:

$$
\mathbf{SF}[\![UT(Cus)]\!] \in \mathbf{SF}[\![UT(\phi_{Cus})]\!] \Leftrightarrow UT(\phi_{Cus}) \sqsubseteq_F UT(Cus), \tag{6.17}
$$

$$
\mathbf{SF}[\![UT(Com)]\!] \in \mathbf{SF}[\![UT(\phi_{Com})]\!] \Leftrightarrow UT(\phi_{Com}) \sqsubseteq_F UT(Com), \tag{6.18}
$$

lo que corresponde a la verificación del comportamiento de los procesos Cus y Com que representan a los participantes *Customer* y *Company*, respectivamente, especificado en el CSP, con respecto al dominio semántico de fallos, asegurando así que se cumplen las propiedades de vivacidad.

De acuerdo con el concepto de refinamiento hacia el tiempo (ver sección 2.2.3.3), la descripción de un proceso no temporizado fija restricciones en el orden y la disponibilidad final de los eventos, y permite que todos los comportamientos temporales sean consistentes con su descripción. En este sentido, sobre la base del trabajo realizado en los párrafos anteriores, establecemos las siguientes relaciones:

$$T(\phi_{Cus}) \sqsubseteq_T Cus \Rightarrow UT(\phi_{Cus}) \sqsubseteq_T UT(Cus), \tag{6.19}$$
$$T(\phi_{Com}) \sqsubseteq_T Com \Rightarrow UT(\phi_{Com}) \sqsubseteq_T UT(Com), \tag{6.20}$$
$$T(\phi_{Cus}) \sqsubseteq_F Cus \Rightarrow UT(\phi_{Cus}) \sqsubseteq_F UT(Cus), \tag{6.21}$$
$$T(\phi_{Com}) \sqsubseteq_F Com \Rightarrow UT(\phi_{Com}) \sqsubseteq_F UT(Com), \tag{6.22}$$

que definen que la verificación de los términos no temporizados CSP es una condición necesaria para la verificación de términos temporizados CSP+T. Esto nos permite chequear el comportamiento temporizado de los componentes sobre la base de la secuencia de eventos admitidos por el modelo no temporizado CSP, excluyendo del análisis la secuencia de eventos que puedan no corresponderse con el orden correcto de los eventos, resultantes de la agregación de las restricciones temporales del modelo temporal CSP+T. Esto ayuda a minimizar el problema de explosión de estados que enfrentan las herramientas de MC, ya que trabajan sobre un modelo no temporizado que es más pequeño que el modelo a implementar, y que corresponde directamente con la correcta secuencia de ejecución de eventos del modelo temporizado.

Por último, se obtiene que el comportamiento de los procesos Cus y Com que representan a los participantes *Customer* y *Company*, respectivamente, es correcto; es decir, todos los comportamientos temporizados de los términos de proceso CSP+T son coherentes con su descripción. En otras palabras, el refinamiento hacia el tiempo de términos de proceso CSP+T es compatible con la descripción no temporizada de términos de proceso CSP, y éstos están sujetos a limitaciones adicionales en el comportamiento temporizado de términos de proceso CSP+T. Así, las relaciones (6.1) y (6.2) son verdaderas.

Como puede verse, realizar la verificación de la forma descrita anteriormente resulta en una transformación algebraica muy laboriosa para su aplicación práctica. Por consiguiente, consideramos que la verificación del comportamiento de los procesos que representan el comportamiento de los

participantes de un BPTM se debe realizar utilizando una herramienta de MC. Dado que estamos trabajando con un cálculo de procesos basado en CSP, usamos FDR2. Como se puede observar en la pantalla de FDR2 de la Fig. 6.2, la verificación de cada modelo de los participantes en CSP, COMPANY ($UT(Com)$) y CUSTOMER ($UT(Cus)$), de la realización del BPTM satisface el comportamiento esperado para cada uno, COMP (es decir, $UT(\phi_{Com})$) y CUST (es decir, $UT(\phi_{Cus})$), respectivamente (ver las marcas de comprobación en las filas uno y dos de la Fig. 6.2, respectivamente). Así, el comportamiento de los participantes *Customer* y *Company* especificados en CSP quedan verificados con respecto a los dominios semánticos de las trazas y los fallos, lo cual asegura propiedades de seguridad y vivacidad, respectivamente. Entonces, podemos obtener que el comportamiento de los términos de proceso Cus y Com son correctos; es decir, todos los comportamientos temporizados de los términos de proceso CSP+T son consistentes con su descripción.

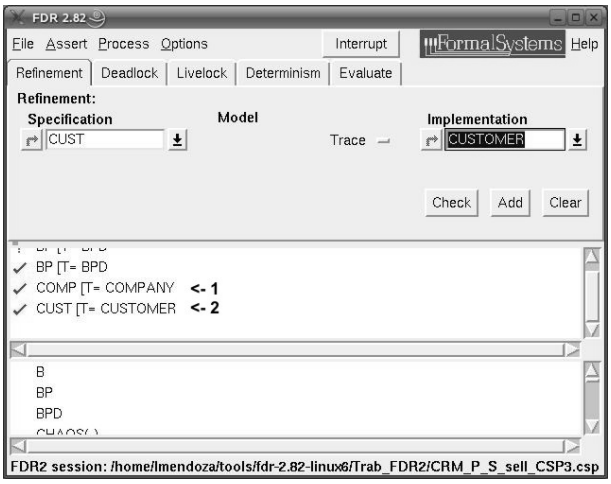


Fig. 6.2 Captura de pantalla de la herramienta FDR2 para el caso CRM

De acuerdo con el Teorema 4.1 (ver sección 4.3, para probar la correctitud de la realización del BPTM con respecto a su comportamiento esperado, se debe demostrar que:

$$PSS \vDash \phi_{PSS} \;\Leftrightarrow\; (Cus \mid [\alpha Cus \| \alpha Com] \mid Com) \backslash \{\mid msg \mid\} \;\vDash\; \phi_{Cus} \wedge \phi_{Com} \,. \tag{6.23}$$

Nosotros hemos verificado con FDR2 y con el proceso algebraico mostrado anteriormente que:

$$Cus \models \phi_{Cus} \;\text{ y }\; Com \models \phi_{Com} \,. \tag{6.24}$$

Sobre la base del diseño detallado de los procesos Cus y Com que representan el comportamiento de los participantes en el BPTM mostrados en la sección 6.1.3, nosotros debemos determinar si estos procesos pueden ser compuestos. Por lo tanto, debemos verificar que ellos cumplen con las siguientes 2 condiciones:

1. Las señales de entrada (Σ_{Cus} y Σ_{Com}), y las señales de salida (Ω_{Cus} y Ω_{Com}) de ambos procesos son disjuntos, lo cual puede ser visto a continuación:

$$\begin{aligned} \Sigma_{Cus} \cap \Sigma_{Com} &= \varnothing \\ \Sigma_{Cus} &= \{msg.cu\_s1.out, msg.cu\_s2.out, msg.cancel.out, msg.cu\_s5.out, \\ &\quad msg.cu\_s6.out\} \\ \Sigma_{Com} &= \{msg.co\_s1.out, msg.co\_s2.out, msg.co\_s3.out, msg.co\_s3.can, \\ &\quad msg.co\_s8.out\} \end{aligned} \tag{6.25}$$

$$\begin{aligned} \Omega_{Cus} \cap \Omega\, Com &= \varnothing \\ \Omega_{Cus} &= \{msg.cu\_s1.in, msg.cu\_s1.last, msg.cu\_s2.in, msg.cu\_s2.last, \\ &\quad msg.cancel.can, msg.cu\_s5.in, msg.cu\_s5.last, msg.cu\_s6.in, \\ &\quad msg.cu\_s6.last\} \\ \Omega_{Com} &= \{msg.co\_s1.in, msg.co\_s1.last, msg.co\_s2.in, msg.co\_s2.last, \\ &\quad msg.co\_s3.in, msg.co\_s3.last, msg.co\_s8.in, msg.co\_s8.last\} \end{aligned} \tag{6.26}$$

2. Los conjuntos de etiquetado de ambos componentes, $\mathbf{L}(Cus)$ y $\mathbf{L}(Com)$, son disjuntos, los cuales también pueden ser verificados como sigue:

$$\begin{aligned} \mathbf{L}(Cus) \cap \mathbf{L}(Com) &= \varnothing \\ \mathbf{L}(Cus) &= \{start.1, cu\_s1, cu\_s2, cu\_s3, cu\_s4, cu\_s5, cu\_s6, xgate.1, \\ &\quad end.1, abort.1\} \\ \mathbf{L}(Com) &= \{start.2, co\_s1, co\_s2, co\_s21, co\_s3, co\_s4, co\_s5, co\_s6, \\ &\quad co\_s7, co\_s8, agate.1, agate.2, end.2, abort.2\} \end{aligned} \tag{6.27}$$

(6.28)

Habiendo verificado que las relaciones (6.25), (6.26), y (6.27), son verdaderas, concluimos que Cus y Com pueden componerse. Por el Lemma 4.1 (ver sección 4.4), tenemos:

$$(Cus \mid [\alpha Cus \| \alpha Com] \mid Com) \backslash \{\mid msg \mid\} \models \phi_{Cus} \wedge \phi_{Com} \tag{6.29}$$

y debido a

$$PSS = (Cus \mid [\alpha Cus \| \alpha Com] \mid Com) \backslash \{\mid msg \mid\} \text{ y } \phi_{PSS} = \phi_{Cus} \wedge \phi_{Com}, \tag{6.30}$$

obtenemos

$$PSS \models \phi_{PSS} \tag{6.31}$$

En conclusión, hemos obtenido la verificación del proceso PSS que representa el comportamiento del BP VPS, a partir de los procesos individuales verificados, Cus y Com, que representan el comportamiento de sus participantes $Customer$ y $Company$. Esto significa que nuestro enfoque ha sido aplicado con éxito a este ejemplo relacionado directamente con la verificación de un BP crítico. También se ha demostrado que el EFVC preserva la composicionalidad de participantes de BP críticos, cuando integramos BPMN como lenguaje de modelado de BP y CSP+T como lenguaje de especificación para la verificación de los procesos. Por último, podemos afirmar que nuestro enfoque puede ser un medio para llevar a cabo la verificación completa de un BP complejo a partir de la definición y verificación de los procesos que representan el conjunto de participantes que intervienen en su realización.

6.2 Red de comunicación de teléfonos móviles

Las compañías de telefonía móvil han tenido un gran auge a nivel mundial en los últimos años. Gran parte del éxito de estas compañías se debe a la alta competitividad que existe entre ellas para atraer nuevos usuarios y ofrecer mejores y nuevos servicios. Es por esta razón que las compañías de telefonía móvil están siempre en la búsqueda de nuevas técnicas, tecnologías y mecanismos que mejoren la calidad de los procesos que soportan sus operaciones, las cuales impactan directamente la satisfacción del cliente. Uno de los BP por medio del cual se puede garantizar el buen funcionamiento de los servicios, es realizar continuamente actividades de monitorización, mantenimiento e instalación de equipos de comunicación móvil.

El caso de aplicación presentado en esta sección se refiere a la supervisión del estado de los dispositivos móviles dentro de las células que constituyen una red de comunicación de teléfonos móviles. Las Estaciones Base de Transmisores (*Base Transceiver Stations*, BTS) contienen un gran número de dispositivos fuertemente interconectados y sus características individuales son susceptibles a reconfiguraciones. Estas características pueden verse afectadas por decisiones acerca de la conveniencia de ofrecer un nuevo servicio, la sustitución de un dispositivo dañado, o simplemente ofrecer un mejor servicio a los clientes. Lograr que el rendimiento de la red sea de

buena calidad requiere garantizar la integridad de la información del estado de cada dispositivo en cada BTS. Un modelo de protocolo de comunicación DDBM con las condiciones de tiempo y restricciones de integridad ha sido elegido para aumentar la velocidad del seguimiento y de las tareas de mantenimiento de las células y el BTS, que permite un acceso más eficiente a la información en cualquier momento. Fig. 6.3 representa un simple, pero conceptualmente relevante, esquema de la situación en la cual se muestran 5 BTS (A to E) y los mensajes que intercambian (*SndMsg* y *AckMsg*) entre ellos.

Cabe indicar que a diferencia del BP analizado en la sección anterior, el cliente de la compañía de telefonía móvil no colabora en la ejecución del BP. En este sentido, este BP es privado [158]; es decir, es de uso exclusivo de la compañía de telefonía móvil que ofrece el servicio, pero tiene un gran impacto en el servicio esperado por el cliente. Es por ello que consideramos que este BP es de alta criticidad para el negocio de la telefonía móvil. Este BP, llamado aquí *Protocolo de comunicación DDBM*, es ejecutado por un sistema de software que denominamos DDBM. Por ello, en este BP los participantes en su ejecución son los componentes de software de DDBM; éstos colaboran concurrentemente para lograr la correcta ejecución del protocolo.

Por otro lado, el modelado y la definición del sistema DDBM es realizado con UML–RT, ya que este lenguaje nos da grandes ventajas para interpretar el sistema, el cual corresponde a la ejecución de procesos concurrentes y de tiempo real. Adicionalmente, tal como mostramos en [38], UML también es usado para especificar y verificar BPs. En síntesis, a través de la aplicación de nuestro enfoque a este caso, se observa que es posible verificar aspectos tanto del sistema de software como del BP que soporta dicho sistema, lo cual reduce enormemente los tiempos del ciclo de desarrollo y redunda en una mayor calidad del sistema obtenido.

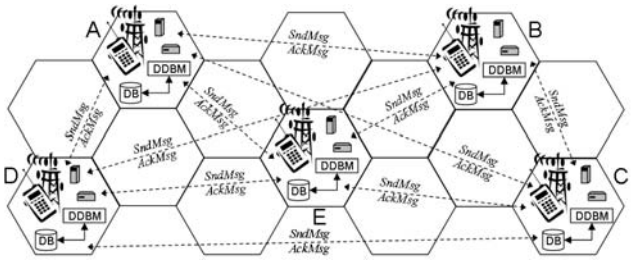


Fig. 6.3 Esquema simplificado de la situación.

Hemos interpretado la solución como un modelo de simulación que permite describir un protocolo de comunicación DDBM con el fin de garantizar

la coherencia de los datos replicados entre las diferentes *Data Base* (DB) locales, de acuerdo a las siguientes condiciones que debe cumplir el protocolo: (a) distribución de réplicas de los datos, (b) integridad de los datos globales, y (c) restricciones temporales. Cada réplica local de datos es actualizada por las distintas $DDBM_i$ (DDBM = $\|_{i:1..n}\ DDBM_i$). La integridad de los datos globales está garantizada mediante el envío de los mensajes apropiados al resto de los $DDBMs$: $SndMsg = \{(i,j) \mid i,j \in \text{DDBM} \wedge i \neq j\}$. Cada $DDBM_i$ actualiza su copia local de los datos globales, y luego debe enviar un mensaje a los otros $DDBM_j$ $(i,j : 1..n,\ i \neq j)$: $AckMsg(s) = \Sigma_{i \in \text{DDBM}-\{j\}} 1'(i,j)$ dentro de un lapso de tiempo fijo.

De acuerdo a esta breve descripción de este ejemplo de aplicación del EFVC, proponemos varias fórmulas CCTL (ver el Apéndice G) que especifican las propiedades derivadas de las necesidades de los usuarios y las reglas de negocio que rigen el funcionamiento del protocolo DDBM global.

Siguiendo los pasos del EFVC, a continuación se muestra cómo verificar el protocolo de comunicación DDBM, que se considera la parte más crítica de cualquier sistema de BTS. Cabe señalar que aunque aquí presentamos una versión simplificada de estos componentes de software, éstos son lo suficientemente claros y completos como para ser considerado un diseño realista y útil de un sistema de gestión de BTS.

6.2.1 Definición y diseño del protocolo de comunicación

Una visión general del funcionamiento del protocolo se muestra en la Fig. 6.4, que representa la sincronización de dos autómatas. Esta sincronización (identificada por paralelogramos) depende de los estados Activo (*Active*) y Pasivo (*Passive*) (señalados con círculos punteados), correspondientes a los dos posibles estados de cada DDBM, de acuerdo con (1) un DDBM solicita una actualización remota de datos (es decir, *Active*), o (2) está *Idle* o respondiendo a una solicitud de actualización de datos global (es decir, *Passive*).

Como resultado de la aplicación de MEDISTAM–RT [21] para modelar el sistema DDBM, cada $DDBM$ está conformado por los componentes $Act\_Control$ y $Man\_Message$ para encapsular el control del sistema, modelado por medio de diagramas de clases y de estructura compuesta UML–RT, como se muestra en la Fig. 6.5. La vista dinámica del sistema es descrita por los dos diagramas UML–TSM, que detallan el intercambio de señales y la sincronización entre los componentes, y entre cada $DDBM$ y su entorno (es decir, los otros $DDBM$ y las DB locales).

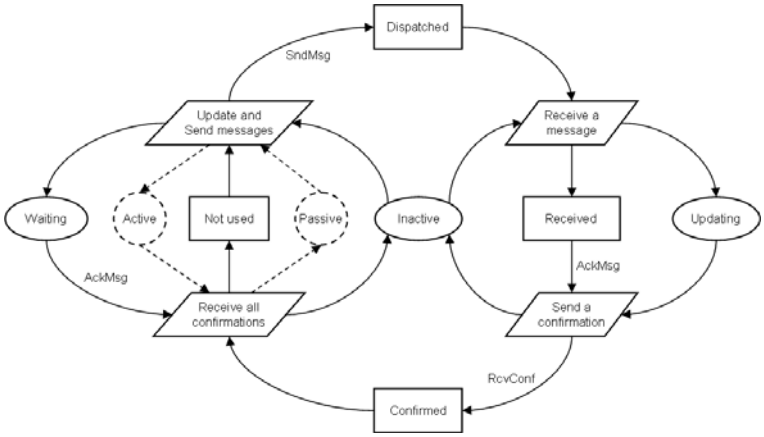


Fig. 6.4 Visión general del funcionamiento del BP Protocolo de comunicación DDBM.

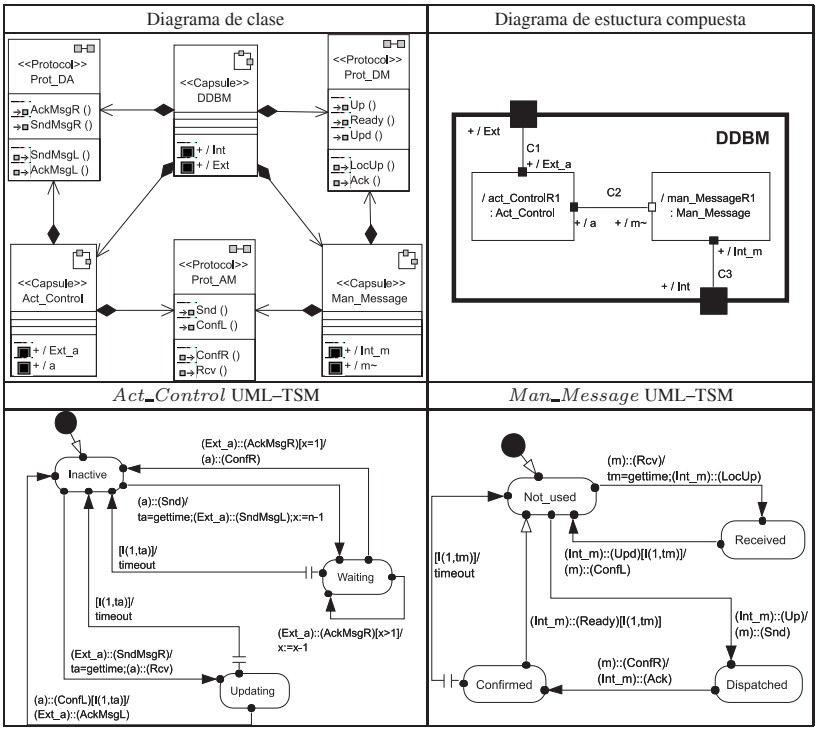


Fig. 6.5 Arquitectura del $DDBM$.

$Act\_Control$ y $Man\_Message$ están defidos en el diseño por diagramas UML–TSM a fin de implementar el protocolo de comunicación que se muestra gráficamente en la Fig. 6.4. Los autómatas descritos con los UML–TSM

representan una simulación del comportamiento observable del protocolo (como se muestra en la Fig. 6.4) y no los componentes del protocolo en sí mismos. Por las características de UML–RT no es posible fusionar diferentes autómatas DDBM en uno solo; es decir, $Act\_Control$ no puede abarcar el comportamiento de cualquier mensaje de transmisión (descrito por $Man\_Message$) en un único autómata. Por lo tanto, es necesario simular el comportamiento combinado de envío de mensajes de actualización (estado *Active*) y recepción de mensajes de actualización (estado *Passive*). Hemos mantenido los estados del componente $Act\_Control$ denotados con óvalos (es decir, *Inactive*, *Waiting*, y *Updating*) y los estados de la transmisión de mensajes en $Man\_Message$ indicados con rectángulos (es decir, *Not_used*, *Dispatched*, *Received*, y *Confirmed*), que corresponden a los estados en la anterior Fig. 6.4. Las transiciones entre los componentes $Act\_Control$ and $Man\_Message$ deben estar de acuerdo con las transiciones denotada por los paralelogramos en la Fig. 6.4, dependiendo de si el componente $DDBM$ se encuentra en el estado *Active* o *Passive* (señalados con círculos punteados) al momento de una actualización, que se consigue con las etiquetas de las transiciones de los dos TSMs (una etiqueta en la acción de una transición provoca el disparo de otra transición que incluye la acción en su guarda).

Por lo tanto, en el protocolo distribuido, los $DDBMs$ aseguran globalmente la integridad de los datos distribuidos entre las n–réplicas. Cuando cualquiera de éstos se actualiza, sólo una de las actualizaciones de datos replicados de los DDBMs es permitida a la vez en cualquier momento.

Como parte del modelado y la definición del protocolo de comunicación DDBM, prestamos especial atención a la inclusión de las restriciones temporales que el protocolo debe cumplir. Por lo tanto, hemos incluido las etiquetas de tiempo necesarias en los UML–TSM de la Fig. 6.5 para asegurar el cumplimiento de los tiempos máximos de espera. Por ejemplo, para garantizar que los componentes $DDBM$ realizan la transición *Send an acknowledge* del protocolo DDBM cuando el componente $Act\_Control$ entra en el estado *Updating* en el momento ta, éstos están comprometidos a recibir —dentro del intervalo de tiempo $[ta, ta + 1]$— la señal *ConfL* enviada por el componente de $Man\_Message$ para cambiar al estado *Inactive*. Si dentro del intervalo de tiempo especificado, el evento *ConfL* no se presenta, un evento *timeout* se levantará después del instante $ta + 1$. Por otra parte, además de la restricción anterior, para evitar un componente $DDBM$ pueda permanecer indefinidamente en la transición *Receive all confirmations* del protocolo DDBM, cuando el componente $Man\_Message$ entra en el estado *Confirmed* en el momento tm, éste debe recibir —dentro del intervalo de tiempo $[tm, tm + 1]$— la señal *Ready* a más tardar en el instante $tm + 1$, de lo contrario se levantará el evento *timeout*.

Mediante la especificación de estos intervalos e instantes de tiempo podemos garantizar que un $DDBM$ no alcance un estado de bloqueo y no se atiendan nuevas actualizaciones. De esta manera el protocolo asegura a nivel global la integridad de datos distribuidos entre las n–réplicas cuando cualquiera de estos se actualiza, de modo que sólo se permite que se lleve a cabo una actualización de datos a la vez en cualquier momento.

Tanto el diseño completo de la $DDBM$ como el diseño detallado de sus componentes se muestran en la Fig. 6.6,, utilizando el lenguaje de especificación formal CSP+T, para realizar la verificación de los componentes individuales, es decir, $Act\_Control$ (identificado con AC) y $Man\_Message$ (identificado con MM), y del diseño del sistema completo (es decir, $DDBM$).

$$
\begin{array}{lll}
T(DDBM) = & \multicolumn{2}{l}{\text{CSP+T}_{g,m}(Act\_Control, C, Man\_Message) =} \\
 & \multicolumn{2}{l}{V_a(T(AC)) \mid [a_{in}, a_{out}] \mid C \mid [m_{in}, m_{out}] \mid V_m(T(MM))} \\
 & & \\
T(AC) = & \multicolumn{2}{l}{t_0.\star \rightarrow Inactive} \\
 & Inactive = & (Ext\_a?SndMsgR \bowtie ta \rightarrow (a!Rcv \rightarrow Updating)) \,\Box \\
 & & (a?Snd \bowtie ta \rightarrow (Ext\_a!SndMsgL \rightarrow Waiting)) \\
 & Updating = & (\text{I}(1, ta).a?ConfL \rightarrow (Ext\_a!AckMsgL \rightarrow Inactive)) \,\Box \\
 & & (\text{I}(1, ta) \rightarrow (timeout \rightarrow Inactive)) \\
 & Waiting = & ((Wait(x) = Ext\_a?AckMsgR \rightarrow Wait(x-1) \text{ if } x > 0) \,\Box \\
 & & (Wait(0) = a!ConfR \rightarrow Inactive)) \,\Box \\
 & & (\text{I}(2, ta) \rightarrow (timeout \rightarrow Inactive)) \\
 & & (x \text{ corresponds to } (n-1)\text{-replicas of data update}) \\
 & \multicolumn{2}{l}{V_{Ext\_a_{in}}(T(AC)) = \{t_0.\star,\ SndMsgR,\ \langle AckMsgR\rangle^{n-1} \mid n \in \mathbb{N}\}} \\
 & \multicolumn{2}{l}{V_{Ext\_a_{out}}(T(AC)) = \{SndMsgL,\ AckMsgL\}} \\
 & \multicolumn{2}{l}{V_{a_{in}}(T(AC)) = \{t_0.\star\ Snd,\ \text{I}(1, ta).ConfL\}} \\
 & \multicolumn{2}{l}{V_{a_{out}}(T(AC)) = \{Rcv,\ ConfR\}} \\
 & & \\
T(MM) = & \multicolumn{2}{l}{t_0.\star \rightarrow Not\_used} \\
 & Not\_used = & (m?Rcv \rightarrow (Int\_m!LocUp \rightarrow Received)) \,\Box \\
 & & (Int\_m?Up \rightarrow (m!Snd \rightarrow Dispatched)) \\
 & Received = & Int\_m?Upd \rightarrow (m!ConfL \rightarrow Not\_used) \\
 & Dispatched = & m?ConfR \bowtie tm \rightarrow (Int\_m!Ack \rightarrow Confirmed) \\
 & Confirmed = & (\text{I}(1, tm).Int\_m?Ready \rightarrow Not\_used) \,\Box \\
 & & (\text{I}(1, tm) \rightarrow (timeout \rightarrow Not\_used)) \\
 & \multicolumn{2}{l}{V_{Int\_m_{in}}(T(MM)) = \{t_0.\star,\ Up,\ Upd,\ \text{I}(1, tm).Ready\}} \\
 & \multicolumn{2}{l}{V_{Int\_m_{out}}(T(MM)) = \{LocUp,\ Ack\}} \\
 & \multicolumn{2}{l}{V_{m_{in}}(T(MM)) = \{t_0.\star,\ Rcv,\ ConfR\}} \\
 & \multicolumn{2}{l}{V_{m_{out}}(T(MM)) = \{Snd,\ ConfL\}}
\end{array}
$$

Fig. 6.6 Especificación del comportamiento de $DDBM$, $Act\_Control$ (AC) y $Man\_Message$ (MM), en términos de CSP+T.

Cada término de proceso CSP+T en la Fig. 6.6 detalla la secuencia de los eventos y acciones que corresponde a las transiciones de los UML–TSM (ver Fig. 6.5) para cambiar de un estado a otro. Los términos de proceso también especifican los intervalos de activación que corresponden a las restricciones de tiempo que los componentes $Act\_Control$ y $Man\_Message$ deben cumplir para lograr el correcto funcionamiento del protocolo de comunicación DDBM. Estos términos serán verificadas respecto de las propiedades que se detallan en la siguiente sección.

6.2.2 Especificación de propiedades

Las Tablas 6.3 y 6.4 muestran la interpretación de la propiedad abstracta ϕ_1 (ver Apéndice G), es decir, que expresa *la garantía del procesamiento de un mensaje a la vez* de acuerdo con los estados *Active* y *Passive* del DDBM, respectivamente. Así, por ejemplo, cuando un DDBM entra en el estado *Active* (es decir, *solicita una actualización remota*), el componente $Act\_Control$ debe realizar la siguiente secuencia de eventos: $\langle Snd, SndMsgL, \langle AckMsgR\rangle^{n-1} \mid n \in \mathbb{N}, ConfR\rangle$; mientras que el componente $Man\_message$ de cada DDBM remoto debe realizar la secuencia de eventos: $\langle Up, Snd, ConfR, Ack, Ready\rangle$ (ver Tabla 6.3).

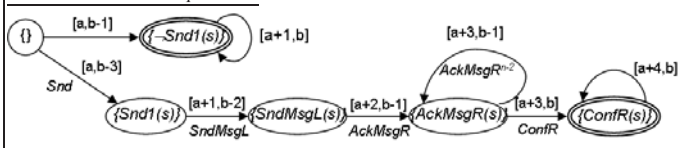
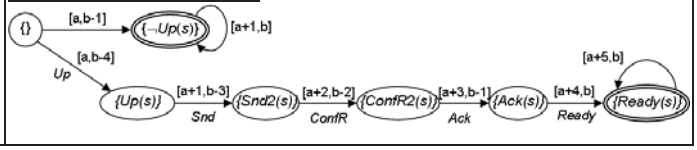
Property	Specification
Solicitud de actualización remota por $Act\_Control$	Fórmula: $\phi_{RUAC} := \mathsf{AG}_{[a,b]}(Snd1(s) \rightarrow \mathsf{A}[SndMsgL(s)\ \mathsf{U}_{[a+1,b-1]}\ (\bigwedge_{x:1..n-1} AckMsgR(s_x) \wedge \mathsf{A}[\bigwedge_{x:1..n-1} AckMsgR(s_x)\ \mathsf{U}_{[a+1,b]}\ ConfR(s)])])$ TBA semánticamente equivalente: 
Solicitud de actualización remota por $Man\_Message$	Fórmula: $\phi_{RUMM} := \mathsf{AG}_{[a,b]}(Up(s) \rightarrow \mathsf{A}[Snd2(s)\ \mathsf{U}_{[a+1,b-2]}\ (Ack2(s) \wedge \mathsf{A}[Ack2(s)\ \mathsf{U}_{[a+2,b-1]}\ (ConfR(s) \wedge \mathsf{A}[ConfR(s)\ \mathsf{U}_{[a+3,b]}\ Ready(s)])])])$ TBA semánticamente equivalente: 

Tabla 6.3 Propiedades de los componentes según el estado *Active* del protocolo de comunicación.

En contraste, cuando el DDBM entra en el estado *Passive* (es decir, *responde a una solicitud de actualización*), el componente $Act\_Control$ debe realizar la siguiente secuencia de eventos: $\langle SndMsgR,\ Rcv, ConfL, AckMsgL\rangle$; y el componente $Man\_Message$ debe realizar la secuencia de eventos: $\langle Rcv, LocUp, Upd, ConfL\rangle$ (véase la Tabla 6.4). Al igual que para el caso CRM, los TBAs mostrados en las Tablas 6.3 y 6.4 fueron obtenidos como resultado de la aplicación del algoritmo descrito en la sección 3.5.1.

Mediante la aplicación de las reglas de transformación indicadas en la Definición 3.8 (sección 3.5.2), obtenemos la interpretación operacional de los TBAs semánticamente equivalentes a las fórmulas CCTL previamente especificadas, de acuerdo con el cálculo procesos CSP+T, tal como se muestra en las Tablas 6.5 y 6.6.

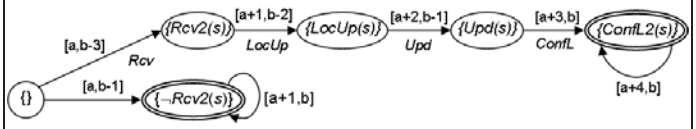
Property	Specification
Respuesta de actualización local por $Act\_Control$	Fórmula: $\phi_{LUAC} := \mathsf{AG}_{[a,b]}(SndMsgR(s) \rightarrow \mathsf{A}[Rcv1(s) \ \mathsf{U}_{[a+1,b-2]} \ (AckMsgL(s) \wedge \mathsf{A}[AckMsgL(s) \ \mathsf{U}_{[a+2,b-1]} \ (ConfL(s) \wedge \mathsf{A}[ConfL(s) \ \mathsf{U}_{[a+3,b]} \ RcvConfL(s)])])])$ TBA semánticamente equivalente: 
Respuesta de actualización local por $Man\_Message$	Fórmula: $\phi_{LUMM} := \mathsf{AG}_{[a,b]}(Rcv2(s) \rightarrow \mathsf{A}[LocUp(s) \ \mathsf{U}_{[a+1,b-1]} \ (Upd(s) \wedge \mathsf{A}[Upd(s) \ \mathsf{U}_{[a+2,b]} \ ConfL(s)])])$ TBA semánticamente equivalente: 

Tabla 6.4 Propiedades de los componentes según el estado *Passive* del protocolo de comunicación.

TBA	CSP+T process term	
ϕ_{LUAC}	$T(\phi_{LUAC}) =$	$t_0.\star \rightarrow T(S_0)$
	$T(S_0) =$	$\mathrm{I}((b-3)-a,a).SndMsgR \rightarrow T(SndMsgR(s))$
	$T(SndMsgR(s)) =$	$(\mathrm{I}((b-2)-(a+1),a+1).Rcv \rightarrow T(Rcv(s))$
	$T(Rcv(s)) =$	$(\mathrm{I}((b-1)-(a+2),a+2).ConfL \rightarrow T(ConfL(s))$
	$T(Conf(s)) =$	$(\mathrm{I}(b-(a+3),a+3).AckMsgL \rightarrow T((AckMsgL(s)))$
	$T(AckMsgL(s)) =$	$SKIP \fatsemi T(\phi_{LUAC})$
ϕ_{LUMM}	$T(\phi_{LUMM}) =$	$t_0.\star \rightarrow T(S_0)$
	$T(S_0) =$	$\mathrm{I}((b-3)-a,a).Rcv \rightarrow T(Rcv(s))$
	$T(Rcv(s)) =$	$(\mathrm{I}((b-2)-(a+1),a+1).LocUp \rightarrow T(LocUp(s))$
	$T(LocUp(s)) =$	$(\mathrm{I}((b-1)-(a+2),a+2).Upd \rightarrow T(Upd(s))$
	$T(Upd(s)) =$	$(\mathrm{I}(b-(a+3),a+3).ConfL \rightarrow T((ConfL(s)))$
	$T(ConfL(s)) =$	$SKIP \fatsemi T(\phi_{LUMM})$

Tabla 6.5 Transformación de los TBAs del estado *Passive* del protocolo de comunicación $DDBM$ al lenguaje de especificación CSP+T.

TBA	CSP+T process term	
ϕ_{RUAC}	$T(\phi_{RUAC}) =$	$t_0.\star \rightarrow T(S_0)$
	$T(S_0) =$	$\mathrm{I}((b-3)-a,a).Snd \rightarrow T(Snd(s))$
	$T(Snd(s)) =$	$(\mathrm{I}((b-2)-(a+1),a+1).SndMsgL \rightarrow T((SndMsgL(s)))$
	$T(SndMsgL(s)) =$	$(\mathrm{I}((b-1)-(a+2),a+2).AckMsgR \rightarrow T((AckMsgR(s)))$
	$T(AckMsgR(s)) =$	$(AM(x) = \mathrm{I}((b-1)-(a+3),a+3).AckMsgR \rightarrow AM(x-1)) \ \Box \ (\mathrm{I}(b-(a+3),a+3).ConfR \rightarrow T(ConfR(s)))$
	$T(ConfR(s)) =$	$SKIP \fatsemi T(\phi_{RUAC})$
ϕ_{RUMM}	$T(\phi_{RUMM}) =$	$t_0.\star \rightarrow T(S_0)$
	$T(S_0) =$	$\mathrm{I}((b-4)-a,a).Up \rightarrow T(Up(s))$
	$T(Up(s)) =$	$(\mathrm{I}((b-3)-(a+1),a+1).Snd \rightarrow T((Snd(s)))$
	$T(Snd(s)) =$	$(\mathrm{I}((b-2)-(a+2),a+2).ConfR \rightarrow T((ConfR(s)))$
	$T(ConfR(s)) =$	$(\mathrm{I}((b-1)-(a+3),a+3).Ack \rightarrow T((Ack(s)))$
	$T(Ack(s)) =$	$(\mathrm{I}(b-(a+4),a+4).Ready \rightarrow T((Ready(s)))$
	$T(Ready(s)) =$	$SKIP \fatsemi T(\phi_{RUMM})$

Tabla 6.6 Transformación de los TBAs del estado *Active* del protocolo de comunicación $DDBM$ al lenguaje de especificación CSP+T.

6.2.3 Verificación de los componentes del sistema y discusión

Obtenidos los términos de proceso CSP+T que representan tanto el modelo del protocolo (ver Fig.6.6) como las propiedades que debe cumplir el modelo (ver Tablas 6.5 y 6.6), seguimos con la verificación de los componentes del sistema. Vamos a verificar que los componentes (es decir, $Act\_Control$ y $Man\_Message$) de la arquitectura $DDBM$ cumple con las propiedades especificadas en la sección 6.2.2. Luego, bajo el dominio semántico del cálculo basado en CSP, se comprueba que se cumplen las siguientes relaciones de refinamiento:

$$T(\phi_{LUAC}) \sqsubseteq_T AC_{LUAC} \ , \ T(\phi_{RUAC}) \sqsubseteq_T AC_{RUAC} \tag{6.32}$$
$$T(\phi_{LUAC}) \sqsubseteq_F AC_{LUAC} \ , \ T(\phi_{RUAC}) \sqsubseteq_F AC_{RUAC} \tag{6.33}$$
$$T(\phi_{LUMM}) \sqsubseteq_T MM_{LUMM} \ , \ T(\phi_{RUMM}) \sqsubseteq_T MM_{RUMM} \tag{6.34}$$
$$T(\phi_{LUMM}) \sqsubseteq_F MM_{LUMM} \ , \ T(\phi_{RUMM}) \sqsubseteq_F MM_{RUMM} \tag{6.35}$$

Siguiendo el proceso que ya mostramos para el caso CRM, vamos a trabajar bajo el modelo semántico no temporizado de CSP y luego usaremos los resultados en el análisis temporizado. A continuación se muestra la verificación del componente $Man\_Message$ (MM) con respecto a las propiedades ϕ_{LUMM} y ϕ_{RUMM}. La Fig. 6.7 muestra los términos de proceso CSP $UT(MM)$, $UT(\phi_{LUMM})$, y $UT(\phi_{RUMM})$, que corresponden al componente de MM y a las propiedades ϕ_{LUMM} y ϕ_{RUMM}, respectivamente.

Vamos a utilizar las Definiciones 2.14 y 2.15 (ver sección 2.2.3.3) para comprobar que:

$$UT(\phi_{LUMM}) \sqsubseteq_t T(\phi_{LUMM}) \ , \ UT(\phi_{RUMM}) \sqsubseteq_t T(\phi_{RUMM}) \tag{6.36}$$
$$UT(MM_{LUMM}) \sqsubseteq_t MM_{LUMM} \ , \ UT(MM_{RUMM}) \sqsubseteq_t MM_{RUMM} \tag{6.37}$$
$$UT(\phi_{LUMM}) \sqsubseteq_f T(\phi_{LUMM}) \ , \ UT(\phi_{RUMM}) \sqsubseteq_f T(\phi_{RUMM}) \tag{6.38}$$
$$UT(MM_{LUMM}) \sqsubseteq_f MM_{LUMM} \ , \ UT(MM_{RUMM}) \sqsubseteq_f MM_{RUMM} \tag{6.39}$$

A través del desdoblado de la definición de cada proceso, tal como mostramos en el caso CRM, demostraremos las relaciones (6.32) a (6.39). El desarrollo de los términos $T(MM)$ y $UT(MM)$ cuando comprobamos el comportamiento de los estados *Passive* y *Active* de un DDBM, denotado MM_{LUMM}, MM_{RUMM}, $UT(MM_{LUMM})$ y $UT(MM_{RUMM})$, respectivamente, nos lleva a la siguiente expresión:

$$MM_{LUMM} = t_0.\star \rightarrow Rcv \rightarrow LocUp \rightarrow Upd \rightarrow ConfL \rightarrow MM_{LUMM}$$
$$MM_{RUMM} = t_0.\star \rightarrow Up \rightarrow Snd \rightarrow ConfR \bowtie tm \rightarrow Ack \rightarrow$$

$$
\begin{array}{lll}
UT(AC) = & \star \rightarrow Inactive & \\
& Inactive = & (Ext\_a?SndMsgR \rightarrow (a!Rcv \rightarrow Updating)) \,\Box \\
& & (a?Snd \rightarrow (Ext\_a!SndMsgL \rightarrow Waiting)) \\
& Updating = & a?ConfL \rightarrow (Ext\_a!AckMsgL \rightarrow Inactive) \\
& Waiting = & ((\, Wait(x) = Ext\_a?AckMsgR \rightarrow Wait(x-1) \text{ if } x > 0) \,\Box \\
& & Wait(0) = a!ConfR \rightarrow Inactive \\
& & (x \text{ corresponds to } (n-1)\text{-replicas of data update}) \\
& V_{Ext\_a_{in}}(T(AC)) = \{SndMsgR,\ \langle AckMsgR\rangle^{n-1} \mid n \in \mathbb{N}\} & \\
& V_{Ext\_a_{out}}(T(AC)) = \{SndMsgL,\ AckMsgL\} & \\
& V_{a_{in}}(T(AC)) = \{Snd,\ ConfL\} & \\
& V_{a_{out}}(T(AC)) = \{Rcv,\ ConfR\} &
\end{array}
$$

$$
\begin{array}{lll}
UT(MM) = & \star \rightarrow Not\_used & \\
& Not\_used = & (m?Rcv \rightarrow (Int\_m!LocUp \rightarrow Received)) \,\Box \\
& & (Int\_m?Up \rightarrow (m!Snd \rightarrow Dispatched)) \\
& Received = & Int\_m?Upd \rightarrow (m!ConfL \rightarrow Not\_used) \\
& Dispatched = & m?ConfR \rightarrow (Int\_m!Ack \rightarrow Confirmed) \\
& Confirmed = & Int\_m?Ready \rightarrow Not\_used \\
& V_{Int\_m_{in}}(UT(MM)) = \{\star,\ Up,\ Upd,\ Ready\} & \\
& V_{Int\_m_{out}}(UT(MM)) = \{LocUp,\ Ack\} & \\
& V_{m_{in}}(UT(MM)) = \{\star,\ Rcv,\ ConfR\} & \\
& V_{m_{out}}(UT(MM)) = \{Snd,\ ConfL\} &
\end{array}
$$

$$
\begin{array}{lll}
UT(\phi_{LUMM}) = & \star \rightarrow UT(S_0) & \\
& UT(S_0) = & Rcv \rightarrow UT(Rcv(s)) \\
& UT(Rcv(s)) = & LocUp \rightarrow UT(LocUp(s)) \\
& UT(LocUp(s)) = & Upd \rightarrow UT(Upd(s)) \\
& UT(Upd(s)) = & ConfL \rightarrow UT(ConfL(s)) \\
& UT(ConfL(s)) = & SKIP \fatsemi UT(\phi_{LUMM})
\end{array}
$$

$$
\begin{array}{lll}
UT(\phi_{RUMM}) = & \star \rightarrow UT(S_0) & \\
& UT(S_0) = & Up \rightarrow UT(Up(s)) \\
& UT(Up(s)) = & Snd \rightarrow UT((Snd(s)) \\
& UT(Snd(s)) = & ConfR \rightarrow UT((ConfR(s)) \\
& UT(ConfR(s)) = & Ack \rightarrow UT((Ack(s)) \\
& UT(Ack(s)) = & Ready \rightarrow UT((Ready(s)) \\
& UT(Ready(s)) = & SKIP \fatsemi UT(\phi_{RUMM})
\end{array}
$$

Fig. 6.7 Especificación del comportamiento de $DDBM$, $Act\_Control$ (AC) y $Man\_Message$ (MM), en términos de CSP.

$$
\begin{pmatrix} \mathrm{I}(1,tm).Ready \rightarrow MM_{RUMM} \\ \Box \\ \mathrm{I}(1,tm) \rightarrow timeout \rightarrow MM_{RUMM} \end{pmatrix}
$$

$$
\begin{aligned}
UT(MM_{LUMM}) &= \star \rightarrow Rcv \rightarrow LocUp \rightarrow Upd \rightarrow ConfL \rightarrow MM_{LUMM} \\
UT(MM_{RUMM}) &= \star \rightarrow Up \rightarrow Snd \rightarrow ConfR \rightarrow Ack \rightarrow Ready \rightarrow MM_{RUMM}
\end{aligned}
$$

El desarrollo de los términos $T(\phi_{LUMM})$, $T(\phi_{RUMM})$, $UT(\phi_{LUMM})$ y $UT(\phi_{RUMM})$ se convierte en:

$$
\begin{aligned}
T(\phi_{LUMM}) = \; & t_0.\star \rightarrow \mathrm{I}((b-3)-a,a).Rcv \rightarrow \mathrm{I}((b-2)-(a+1),a+1).LocUp \rightarrow \\
& \mathrm{I}(((b-1)-(a+2),a+2).Upd \rightarrow \mathrm{I}(b-(a+3),a+3).ConfL \rightarrow \\
& T(\phi_{LUMM})
\end{aligned}
$$

$$
\begin{aligned}
T(\phi_{RUMM}) = \; & t_0.\star \rightarrow \mathrm{I}((b-4)-a,a).Up \rightarrow \mathrm{I}((b-3)-(a+1),a+1).Snd \rightarrow \\
& \mathrm{I}((b-2)-(a+2),a+2).ConfR \rightarrow \mathrm{I}((b-1)-(a+3),a+3).Ack \rightarrow \\
& \mathrm{I}(b-(a+4),a+4).Ready \rightarrow T(\phi_{LUMM})
\end{aligned}
$$

$$UT(\phi_{LUMM}) = \star \rightarrow Rcv \rightarrow LocUp \rightarrow Upd \rightarrow ConfL \rightarrow UT(\phi_{LUMM})$$
$$UT(\phi_{RUMM}) = \star \rightarrow Up \rightarrow Snd \rightarrow ConfR \rightarrow Ack \rightarrow Ready \rightarrow UT(\phi_{LUMM})$$

Aplicando la Definición 2.14, podemos realizar la verificación de las relaciones (6.36) y (6.37), según el modelo de trazas no temporizadas de CSP. Sea,

$$\begin{aligned}
strip(T(\phi_{LUMM})) &= \langle \star, Rcv, LocUp, Upd, ConfL \rangle \\
strip(T(\phi_{RUMM})) &= \langle \star, Up, Snd, ConfR, Ack, Ready \rangle \\
strip(MM_{LUMM}) &= \langle \star, Rcv, LocUp, Upd, ConfL \rangle \\
strip(MM_{RUMM}) &= \langle \star, Up, Snd, ConfR, Ack, Ready \rangle \\
\mathsf{traces}(UT(\phi_{LUMM})) &= \{\langle\rangle, \langle \star, Rcv, LocUp, Upd, ConfL \rangle\} && (6.40)\\
\mathsf{traces}(UT(\phi_{RUMM})) &= \{\langle\rangle, \langle \star, Up, Snd, ConfR, Ack, Ready \rangle\} && (6.41)\\
\mathsf{traces}(UT(MM_{LUMM})) &= \{\langle\rangle, \langle \star, Rcv, LocUp, Upd, ConfL \rangle\} && (6.42)\\
\mathsf{traces}(UT(MM_{RUMM})) &= \{\langle\rangle, \langle \star, Up, Snd, ConfR, Ack, Ready \rangle\}\,, && (6.43)
\end{aligned}$$

escribimos:

$$\begin{aligned}
strip(T(\phi_{LUMM})) \in \mathsf{traces}(UT(\phi_{LUMM})) &\Leftrightarrow UT(\phi_{LUMM}) \sqsubseteq_t T(\phi_{LUMM}),\\
strip(T(\phi_{RUMM})) \in \mathsf{traces}(UT(\phi_{RUMM})) &\Leftrightarrow UT(\phi_{RUMM}) \sqsubseteq_t T(\phi_{RUMM}),\\
strip(MM_{LUMM}) \in \mathsf{traces}(UT(MM_{LUMM})) &\Leftrightarrow UT(MM_{LUMM}) \sqsubseteq_t MM_{LUMM},\\
strip(MM_{RUMM}) \in \mathsf{traces}(UT(MM_{RUMM})) &\Leftrightarrow UT(MM_{RUMM}) \sqsubseteq_t MM_{RUMM},
\end{aligned}$$

que corresponden a las relaciones (6.36) y (6.37). Esto significa que los términos de proceso CSP+T, tanto del modelo del protocolo como de las propiedades esperadas, son un refinamiento hacia trazas temporizadas de los términos de proceso CSP propuestos.

Aplicando la Definición 2.15, podemos realizar la verificación de las relaciones (6.38) y (6.39), según el modelo de fallos no temporizado de CSP. Sea,

$$\begin{aligned}
\mathbf{SF}[\![T(\phi_{LUMM})]\!] &= \{(\langle\star\rangle, \{Rcv\}), (\langle \star, Rcv, LocUp, Upd, ConfL\rangle, \{\})\}\\
\mathbf{SF}[\![T(\phi_{RUMM})]\!] &= \{(\langle\star\rangle, \{Up\}), (\langle \star, Up, Snd, ConfR, Ack, Ready\rangle, \{\})\}\\
\mathbf{SF}[\![MM_{LUMM}]\!] &= \{(\langle\star\rangle, \{Rcv\}), (\langle \star, Rcv, LocUp, Upd, ConfL\rangle, \{\})\}\\
\mathbf{SF}[\![MM_{RUMM}]\!] &= \{(\langle\star\rangle, \{Up\}), (\langle \star, Up, Snd, ConfR, Ack, Ready\rangle, \{\}),\\
&\quad\ (\langle \star, Up, Snd, ConfR, Ack\rangle, \{Ready\})\}\\
\mathbf{SF}[\![UT(\phi_{LUMM})]\!] &= \{(\langle\star\rangle, \{Rcv\}), (\langle \star, Rcv, LocUp, Upd, ConfL\rangle, \{\})\} && (6.44)\\
\mathbf{SF}[\![UT(\phi_{RUMM})]\!] &= \{(\langle\star\rangle, \{Up\}), (\langle \star, Up, Snd, ConfR, Ack, Ready\rangle, \{\})\} && (6.45)\\
\mathbf{SF}[\![UT(MM_{LUMM})]\!] &= \{(\langle\star\rangle, \{Rcv\}), (\langle \star, Rcv, LocUp, Upd, ConfL\rangle, \{\})\} && (6.46)\\
\mathbf{SF}[\![UT(MM_{RUMM})]\!] &= \{(\langle\star\rangle, \{Up\}), (\langle \star, Up, Snd, ConfR, Ack, Ready\rangle, \{\}), && (6.47)\\
&\quad\ (\langle \star, Up, Snd, ConfR, Ack\rangle, \{Ready\})\}
\end{aligned}$$

escribimos:

$$\begin{aligned}
\mathbf{SF}[\![\, T(\phi_{LUMM})]\!] \in \mathbf{SF}[\![\, UT(\phi_{LUMM})]\!] &\Leftrightarrow\ UT(\phi_{LUMM}) \sqsubseteq_f\ T(\phi_{LUMM}) \\
\mathbf{SF}[\![\, T(\phi_{RUMM})]\!] \in \mathbf{SF}[\![\, UT(\phi_{RUMM})]\!] &\Leftrightarrow\ UT(\phi_{RUMM}) \sqsubseteq_f\ T(\phi_{RUMM}), \\
\mathbf{SF}[\![\, MM_{LUMM}]\!] \in \mathbf{SF}[\![\, UT(MM_{LUMM})]\!] &\Leftrightarrow\ UT(MM_{LUMM}) \sqsubseteq_f\ MM_{LUMM} \\
\mathbf{SF}[\![\, MM_{RUMM}]\!] \in \mathbf{SF}[\![\, UT(MM_{RUMM})]\!] &\Leftrightarrow\ UT(MM_{RUMM}) \sqsubseteq_f\ MM_{RUMM},
\end{aligned}$$

que corresponden a las relaciones (6.38) y (6.39). Esto significa que los términos de proceso CSP+T, tanto del modelo del protocolo como de las propiedades esperadas, son un refinamiento hacia fallos temporizados de los términos de proceso CSP propuestos. Además, por las relaciones (6.40) a (6.43), escribimos:

$$\mathsf{traces}(\, UT(MM_{LUMM})) \in \mathsf{traces}(\, UT(\phi_{LUMM})) \ \Leftrightarrow\ UT(\phi_{LUMM}) \sqsubseteq_T\ UT(MM_{LUMM}), \tag{6.48}$$
$$\mathsf{traces}(\, UT(MM_{RUMM})) \in \mathsf{traces}(\, UT(\phi_{RUMM})) \ \Leftrightarrow\ UT(\phi_{RUMM}) \sqsubseteq_T\ UT(MM_{RUMM}), \tag{6.49}$$

lo que corresponde a la verificación del comportamiento especificado de $Man\_Message$ en términos de CSP (es decir, $UT(MM_{LUMM})$ y $UT\ (MM_{RUMM})$) de acuerdo con el comportamiento esperado para este componente, también especificada en CSP (es decir, $UT(\phi_{LUMM})$ y $UT(\phi_{RUMM})$). Como resultado, el comportamiento del componente $Man\_Message$ especificado en CSP fue verificado con respeto al campo semántico de trazas, lo que asegura que se cumplen las propiedades de seguridad.

También, por las relaciones (6.44) a (6.47), escribimos:

$$\mathbf{SF}[\![\, UT(MM_{LUMM})]\!] \in \mathbf{SF}[\![\, UT(\phi_{LUMM})]\!] \ \Leftrightarrow\ UT(\phi_{LUMM}) \sqsubseteq_F\ UT(MM_{LUMM}), \tag{6.50}$$
$$\mathbf{SF}[\![\, UT(MM_{RUMM})]\!] \in \mathbf{SF}[\![\, UT(\phi_{RUMM})]\!] \ \Leftrightarrow\ UT(\phi_{RUMM}) \sqsubseteq_F\ UT(MM_{RUMM}), \tag{6.51}$$

lo que corresponde a la verificación del comportamiento del componente $Man\_Message$, especificado en el CSP, con respecto al dominio semántico de fallos, asegurando así que se cumplen las propiedades de vivacidad.

De acuerdo con el concepto de refinamiento hacia el tiempo que describimos en el caso CRM, establecemos las siguientes relaciones:

$$T(\phi_{LUMM}) \sqsubseteq_T MM_{LUMM}\ \Rightarrow\ UT(\phi_{LUMM}) \sqsubseteq_T\ UT(MM_{LUMM}), \tag{6.52}$$
$$T(\phi_{RUMM}) \sqsubseteq_T MM_{RUMM}\ \Rightarrow\ UT(\phi_{RUMM}) \sqsubseteq_T\ UT(MM_{RUMM}), \tag{6.53}$$
$$T(\phi_{LUMM}) \sqsubseteq_F MM_{LUMM}\ \Rightarrow\ UT(\phi_{LUMM}) \sqsubseteq_F\ UT(MM_{LUMM}), \tag{6.54}$$
$$T(\phi_{RUMM}) \sqsubseteq_F MM_{RUMM}\ \Rightarrow\ UT(\phi_{RUMM}) \sqsubseteq_F\ UT(MM_{RUMM}), \tag{6.55}$$

De forma análoga a la descripción que hicimos para el caso CRM, se obtiene que el comportamiento del componente $Man\_Message$ es correcto. Así, las relaciones (6.32) a (6.35) son verdaderas. Usando nuevamente la herramienta de MC FDR2, observamos en la captura de pantalla de la Fig. 6.8, la verificación de cada modelo de componentes en CSP, $Act\_Control$ (es decir, $UT(AC)$) y $Man\_Message$ (es decir, $UT(MM)$), de la implementación del protocolo de comunicación $DDBM$ satisface el comportamiento esperado para cada uno, $ESP\_Act\_Control$ (es decir, $UT(\phi_{LUAC}) \| UT(\phi_{RUAC})$) y $ESP\_Man\_Message$ (es decir, $UT(\phi_{LUMM}) \| UT(\phi_{RUMM})$), respectivamente (ver las marcas de comprobación en las filas uno y dos, respectivamente, de la Fig. 6.8).

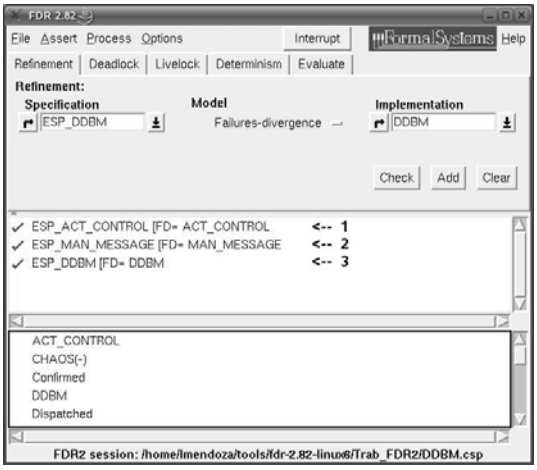


Fig. 6.8 Captura de pantalla de la herramienta FDR2 para el caso DDBM

De acuerdo con el Teorema 4.1 (ver sección 4.3), para probar la correctitud de la implementación del componente $DDBM$ con respecto a su comportamiento esperado, se debe demostrar que:

$$DDBM \vDash \phi_{DDBM} \;\Leftrightarrow\; Act\_Control \| Man\_Message \vDash \phi_{Act\_Control} \wedge \phi_{Man\_Message} \,. \tag{6.56}$$

Nosotros hemos verificado con FDR2 que:

$$Act\_Control \vDash \phi_{Act\_Control} \text{ y } Man\_Message \vDash \phi_{Man\_Message} \,. \tag{6.57}$$

Sobre la base del diseño detallado de los componentes $Act\_Control$ y $Man\_Message$ mostrados en la Fig. 6.6, nosotros debemos determinar si

estos componentes pueden ser compuestos. Por lo tanto, debemos verificar que ellos cumplen con las siguientes 2 condiciones:

1. Las señales de entrada (Σ_{AC} y Σ_{MM}), y las señales de salida (Ω_{AC} y Ω_{MM}) de ambos componentes son disjuntos, lo cual puede ser visto a continuación:

$$\begin{aligned} \Sigma_{AC} \cap \Sigma_{MM} &= \varnothing \\ \Sigma_{AC} &= V_{Ext\_a_{in}}(T(AC)) \cup V_{a_{in}}(T(AC)) = \{SndMsgR,\ \langle AckMsgR\rangle^{n-1} \mid n \in \mathbb{N},\ Snd,\ ConfL\} \\ \Sigma_{MM} &= V_{Int\_m_{in}}(T(MM)) \cup V_{m_{in}}(T(MM)) = \{Up,\ Upd,\ Ready,\ Rcv,\ ConfR\} \end{aligned} \tag{6.58}$$

$$\begin{aligned} \Omega_{A\_C} \cap \Omega M\_M &= \varnothing \\ \Omega_{AC} &= V_{Ext\_a_{out}}(T(AC)) \cup V_{a_{out}}(T(AC)) = \{SndMsgL,\ AckMsgL, Rcv,\ ConfR\} \\ \Omega_{MM} &= V_{Int\_m_{out}}(T(MM)) \cup V_{m_{out}}(T(MM)) = \{LocUp,\ Ack,\ Snd,\ ConfL\} \end{aligned} \tag{6.59}$$

2. Los conjuntos de etiquetado de ambos componentes, $\mathbf{L}(AC)$ y $\mathbf{L}(MM)$, son disjuntos, los cuales también pueden ser verificados como sigue:

$$\begin{aligned} \mathbf{L}(AC) \cap \mathbf{L}(MM) &= \varnothing \\ \mathbf{L}(AC) &= \{SndMsgR,\ Rcv1,\ ConfL1,\ AckMsgL,\ Snd1,\ SndMsgL,\ AckMsgR,\ ConfR1\} \\ \mathbf{L}(MM) &= \{Rcv2,\ LocUp,\ Upd,\ ConfL2,\ Up,\ Snd2,\ ConfR2,\ Ack,\ Ready\} \end{aligned} \tag{6.60}$$

Habiendo verificado que las relaciones (6.58), (6.59), y (6.60), son verdaderas, concluimos que $Act\_Control$ y $Man\_Message$ pueden ser compuestos. Por el Lemma 4.1 (véase sección 4.4), tenemos:

$$Act\_Control \| Man\_Message \models \phi_{Act\_Control} \wedge \phi_{Man\_Message} \tag{6.61}$$

y debido a

$$DDBM = Act\_Control \| Man\_Message \text{ and } \phi_{DDBM} = \phi_{Act\_Control} \wedge \phi_{Man\_Message}, \tag{6.62}$$

obtenemos

$$DDBM \models \phi_{DDBM} \tag{6.63}$$

Hemos demostrado que podemos obtener la verificación del $DDBM$ a partir de sus componentes individuales verificados, $Act\_Control$ y $Man\_Message$. En este sentido, hemos podido aplicar nuestro enfoque a este ejemplo, corroborando que el EFVC preserva la composicionalidad de com-

ponentes de sistemas de software que soportan BP críticos como resultado de un proceso de refinamiento de la fase de diseño del sistema. En síntesis, el EFVC puede ser utilizado para soportar la verificación de software que implementa procesos complejos con criticidad, a partir de la verificación individual de los procesos que modelan el comportamiento de colecciones de componentes.

En este capítulo se mostró la aplicación de nuestro enfoque a dos ejemplos de interés empresarial e industrial, la cual puede ser vista como una demostración de la viabilidad de las ideas expuestas a lo largo de la tesis doctoral. En el próximo capítulo sintetizamos los logros alcanzados a lo largo de esta tesis doctoral y enumeramos las próximas acciones que realizaremos para darle continuidad a este trabajo.

LOGROS OBTENIDOS

Capítulo 7
Discusión de resultados y conclusiones

Resumen El objeto de este capítulo es enumerar los principales resultados y conclusiones obtenidos a lo largo del desarrollo de la tesis doctoral. En la sección 7.1 sintetizamos los resultados más importantes y las principales contribuciones que se han obtenido a lo largo de este trabajo. Luego, en la sección 7.2 indicamos las conclusiones más importantes de la tesis doctoral. Finalmente, en la sección 7.3, se hace una propuesta del trabajo futuro que permite darle continuidad a la línea de investigación que se inicia con este trabajo doctoral.

7.1 Resultados – Principales contribuciones

El logro de los objetivos indicados en el capítulo 1 nos permite resumir las principales contribuciones de la tesis doctoral como sigue:

1. *Propuesta de una herramienta metodológica de modelado formal de BP, así como la especificación formal de las propiedades que debe satisfacer*. El desarrollo de este trabajo nos indica que tanto para el diseño de SCS como para el diseño de BP con criticidad, es tan importante la especificación y la verificación de los requisitos funcionales (*qué debe hacer* el SCS o el BP) como de los requisitos no funcionales (*cómo de bien debe hacerse*) relacionados con el cumplimiento de restricciones temporales.
2. *Definición de una notación formal basada en un cálculo de procesos para poder llevar a cabo una verificación del modelo del BP respecto de las satisfacción de sus propiedades*. Son pocos los trabajos hallados en la literatura referidos al aprovechamiento de las fortalezas de los cálculos de proceso en el ámbito de los BP que permitan no sólo verificar aspectos del control de flujo, sino también aspectos de composición de los procesos que representan el comportamiento de los participantes en un BP.
3. *Definición de una semántica formal para las construcciones temporales, concurrentes y de comunicación, de (un subconjunto de) los elementos notacionales de BPMN*. Con esta propuesta estamos incorporando a la especificación, el modelado y la verificación de BP, aspectos como la duración de las tareas, tiempos de espera o de expiración, que no había sido

tomado en cuenta con anterioridad en la especificación, el modelado y la verificación de BP.

4. *Propuesta de un entorno/infraestructura de verificación composicional de BP que permite la implementación de herramientas software para la transformación de los modelos de BP en especificaciones formales, así como la expresión formal de propiedades no funcionales y requisitos temporales*. Tradicionalmente, la incorporación del tiempo como objeto de verificación torna complejo el proceso de verificación de SCS complejos, el cual puede ser utilizado en la verificación de BP con criticidad. En consecuencia, hemos propuesto el EFVC aplicable a SCS complejos, el cual puede ser utilizado en la verificación de BP con criticidad, que explota las fortalezas del cálculo de procesos CSP y la notación CSP+T, la cual está basada en CSP.
5. *Integración de distintos formalismos que facilitan la especificación, el modelado formal y la verificación composicional del cumplimiento de requisitos no funcionales tanto de BP con criticidad como de SCS complejos*. A través de la transformación de los distintos formalismos utilizados para la conformación de la herramienta metodológica al dominio semántico soportado por el cálculo de procesos CSP, se logra la conducción del proceso de verificación y la formalización del EFVC como elemento fundamental de la propuesta.
6. *Aplicación de la herramienta metodológica propuesta a dos ejemplos de interés empresarial e industrial que permiter probar su viabilidad tanto para BPs con criticidad como para SCS complejos*. El primer caso se refiere a un BP considerado crítico para la estrategia CRM, mientras que el segundo corresponde al sistema que implementa el proceso crítico del protocolo de comunicación que constituye el corazón de una red de comunicación de teléfonos móviles. Adicionalmente, se muestra cómo nuestra propuesta puede ser integrada con otros enfoques metodológicos (en esta ocasión, MEDISTAM–RT) permitiendo la especificación, el modelado y la verificación de propiedades no funcionales y de restricciones de tiempo que caracterizan a cualquier proceso con criticidad.

7.2 Conclusiones

La conclusión fundamental derivada de este trabajo es la demostración de que las técnicas que han sido exitosas en el dominio de la IS, como la verificación automática, pueden ser aplicadas al dominio del BPM. A la fecha, la IS ha logrado consolidar el uso de técnicas para detectar errores en sistemas de software, circuitos electrónicos del hardware y protocolos de comuni-

cación, entre otros. En este sentido, nosotros hemos realizado aportaciones que muestran firmemente que la verificación composicional, soportada por el MC, es aplicable a la disciplina de BPM, con el objeto de detectar errores en las etapas de análisis, diseño y definición de BP; es decir, antes de su implantación en sistemas de uso empresarial o industrial. Con la verificación composicional y el MC podemos demostrar de forma automatizada la presencia o ausencia de determinados tipos de propiedades que sistemas y procesos con criticidad deben cumplir. En particular, aquellas relacionadas con requisitos no funcionales y de rendimiento donde se establezcan restricciones temporales.

Sobre la base de lo anterior, en esta tesis doctoral hemos realizado aportaciones para la especificación, el modelado y la verificación de propiedades no funcionales y temporales de la ejecución de BP, cuyas ideas iniciales provienen de nuestros trabajos en el área de la IS, específicamente en lo relativo a la especificación, el modelado y la verificación de SCS. Teniendo como eje fundamental la técnica de MC, pudimos integrar distintos formalismos en el EFVC que permite tanto la verificación de SCS como de BP con criticidad.

Por otro lado, también pudimos mostrar cómo las fortalezas de los cálculos de proceso, en general, y el cálculo de proceso CSP y su superconjunto CSP+T, en particular, son muy adecuados para la especificación, el modelado y la verificación tanto de SCS como de BP con criticidad. Específicamente para los BP con criticidad, en este trabajo se mostró como CSP+T cubre completamente los aspectos temporales necesarios para precisar la semántica de BPMN. Adicionalmente mostramos que CSP+T también puede ser utilizado para establecer una semántica precisa de UML–RT al momento de diseñar SCS, mediante el uso de MEDISTAM–RT. Y en trabajos nuestros citados a lo largo del tomo se muestra como hemos utilizado CSP+T para verificar BP críticos modelados con el perfil UML EDOC [38, 140] y con BPMN [37, 141, 142]. En síntesis, hemos demostrado la viabilidad del uso de los cálculos de proceso CSP y CSP+T en el ámbito del BPM, los cuales han demostrado ser también exitosos en el ámbito de la IS.

Finalmente, consideramos que las aportaciones en la verificación de requisitos no funcionales y temporales de BP está muy relacionado con la verificación de requisitos no funcionales y temporales de coordinaciones y composiciones de WS. Los WS son dependientes de aspectos de sincronización, donde las restricciones temporales son críticas y el cálculo de procesos CSP+T presenta (tal como lo mostramos en el capítulo 3) una serie de fortalezas para la especificación, el modelado y la verificación de estos aspectos. En consecuencia, las soluciones que se han propuesto en esta tesis doctoral a nivel de colaboración y coordinación de BP podrían ser también extrapolables a la orquestación de WS.

7.3 Trabajo futuro

Nuestra idea fundamental es fortalecer la línea de investigación que se inicia con los resultados de esta tesis doctoral. En este sentido, consideramos en este momento centrarnos en 3 grandes metas iniciales, que permitan lograr a la largo plazo una serie de publicaciones y contribuciones a la comunidad científica y consolidar un cuerpo de conocimiento que ayude a establecer vínculos sólidos entre la IS y el BPM. Estas metas iniciales son:

1. *Aplicar la herramienta metodológica a otros BP con criticidad que sean de interés en el campo industrial con la finalidad de ajustar y probar la capacidad de funcionamiento de la misma*. Adicionalmente, utilizaremos la experiencia y los resultados obtenidos para conducir un proceso de evaluación de la herramienta metodológica según los últimos avances en el área de los métodos de evaluación dentro de la IS. Estimamos orientar el proceso de evaluación utilizando la metodología DESMET [114], la cual nos permite seleccionar el método de evaluación más adecuado según las características actuales de la herramienta metodológica.
2. *Proseguir con la automatización la herramienta metodológica a medio plazo continuando con la implementación del diseño de la herramienta de software propuesta en este trabajo para tal fin* (ver capítulo 5). En paralelo a lo indicado en el inciso anterior, se desarrollarían iterativamente las distintas herramientas que conformarían el workbench que soportaría la aplicación de la herramienta metodológica, bajo situaciones reales de uso, para afinar su funcionamiento y realizar las pruebas que garanticen su calidad.
3. *Utilizar la herramienta metodológica como base para el diseño de una herramienta de software que permita obtener los modelos iniciales del diseño de los sistemas que soportan los BP especificados, modelados y verificados utilizando nuestra propuesta*. Partiendo de (ver Fig. 7.1): (1) la semántica propuesta que permite obtener los términos de proceso CSP+T que definen el comportamiento de BP con criticidad y, (2) las reglas de transformación que construyen los términos de proceso CSP+T a partir de los modelos UML–RT de un sistema; utilizar el dominio semántico de CSP+T como puente entre el modelo de BP bajo BPMN y el modelo inicial en UML del sistema que los implemente (punto (3) de la Fig. 7.1). La meta final a largo plazo es contar con algunos mecanismos sustentados formalmente que faciliten la obtención de modelos iniciales para sistemas en UML a partir de modelos BPMN de BP susceptibles de automatización.

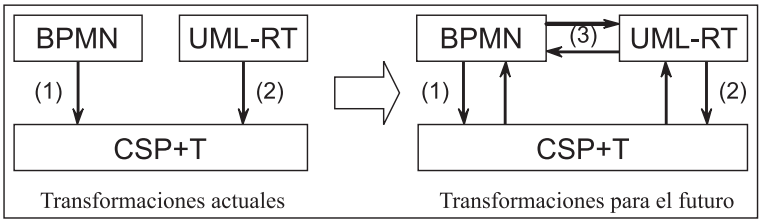


Fig. 7.1 Puente entre BPMN y UML mediante CSP+T.

APÉNDICES

Apéndice A
Visión general de BPMN

En este apéndice se introducen los elementos notacionales generales que conforman el lenguaje BPMN, y que dan un marco conceptual a los elementos notacionales de BPMN que manejan tiempo y que se analizan en la sección 3.3.2.

A.1 Aspectos generales de BPMN en el contexto de la verificación

BPMN es una notación gráfica que describe la lógica de los pasos de un BP o de un WS [60, 216, 216]. Ha sido diseñada especialmente para coordinar la secuencia de los procesos y los mensajes que fluyen entre los participantes de las diferentes actividades dentro de un BP. BPMN es un sistema gráfico de modelado de procesos que representa la culminación de trabajos de consolidación de ideas aportadas por otros estándares anteriores; algunos de los cuales actualmente siguen siendo usados. Ejemplos de estándares que han sido analizados y revisados como base de partida para el desarrollo del BPMN son [158]: diagramas de actividad de UML, UML EDOC, IDEF, ebXML BPSS, ADF, RosettaNet, LOVeM y EPC. La rápida difusión de BPMN lo ha convertido rápidamente en el estándar en el dominio del BPM, hasta el punto de que los fabricantes de herramientas que ofrecen soporte automatizado a la gestión por procesos, han realizado esfuerzos a la readaptación del lenguaje de modelado de sus productos para que soporten BPMN.

BPMN también puede ser soportado por un modelo interno que permite la generación de ejecutables en WS–BPEL, creando así un puente estandarizado para llenar la brecha entre el diseño de BP y la automatización de BP [153]. Según [163], BPMN estructuralmente es un super–conjunto de WS–BPEL. No hay dificultades fundamentales en el mapeo de un proceso WS–BPEL en un diagrama BPMN equivalente. En otras palabras, cualquier proceso WS–BPEL se puede visualizar como un diagrama BPMN sin reorganizar el workflow. Pero no siempre es posible mapear directamente un diagrama BPMN en un proceso WS–BPEL equivalente. Flujos de secuencia arbitrarios permitidos en BPMN son similares a las declaraciones `GOTO` en algunos lenguajes de programación de software. Sin un análisis y re-

definición previo de las estructuras de flujo del diagrama es prácticamente imposible mapear correctamente todos los procesos [163]. Entonces, desde el punto de vista de la verificación, no se puede proceder directamente a verificar un BP modelado con BPMN usando WS–BPEL.

Por otra parte, de acuerdo con [218], la especificación de BPMN no tiene una semántica formal del comportamiento, la cual es necesaria para la especificación y la verificación del comportamiento de actividades de BP críticos. Esto es particularmente importante cuando se especifica la colaboración de BP, donde la coordinación de actividades depende del orden de ejecución y de la duración de cada una. Estamos convencidos de que los analistas y diseñadores de BP necesitan herramientas y enfoques metodológicos que soporten la verificación de BP críticos como parte del BPM. La verificación de BP, sobre todo en etapas tempranas del ciclo de desarrollo, permite tomar acciones correctivas a tiempo y a bajo costo para las empresas. En este sentido, nuestra propuesta soporta a los analistas y diseñadores que trabajan en BPM para conducir la verificación temporal de los modelos de BP críticos, antes de comenzar la fase de implementation del ciclo de vida del software.

A.2 Elementos notacionales – Diagrama de Proceso de Negocio

BPMN permite expresar los BP en un BPD. El BPD se usa para modelar gráficamente las operaciones de los BP, de forma que los usuarios no técnicos del negocio puedan leer y comprender hasta los procesos más complejos. Dentro de un BPD, el usuario crea el modelo de procesos del negocio, representado por una red de objetos gráficos que muestran las actividades y el workflow en orden de ejecución. Los BPD pueden mapear los procesos a los lenguajes de ejecución de BP para automatizarlos usando las notaciones que definen las normas BPMN. En esencia, BPMN se compone de 4 conjuntos de elementos [158]: *Objetos de flujo*, *Objetos de conexión*, *Swinlanes* y *Artefactos*.

A.2.1 Objetos de flujo

Son los elementos gráficos principales utilizados para definir el comportamiento de los procesos; éstos son: *Eventos*, *Actividades* y *Compuertas*. En la Tabla A.1 se describen estos elementos. A su vez, una actividad puede ser

categorizada en *Tarea* o *Sub–proceso*, pero para los efectos de este trabajo esta distinción no es relevante.

Elemento	Descripción	Notación
Evento	Es algo que "sucede" durante la ejecución de un BP. Estos eventos afectan al flujo del proceso y tienen generalmente un disparador (*trigger*) o un impacto. Existen tres tipos de eventos: *inicio*, *intermedio* y *fin*.	
Actividad	Es un término genérico para un trabajo que ejecuta el negocio. Una actividad puede ser atómica (*tarea*) o compuesta (un *BP* o *sub–proceso*).	
Compuerta	Se identifican con un elemento en forma de diamante que representa una *decisión*, *bifurcación* o *unión* de flujo de secuencia dentro del diagrama.	

Tabla A.1 Objetos de flujo medulares de BPMN.

A.2.2 Objetos de conexión

Se encargan de conectar los objetos de flujo. Hay 3 formas de conexión: *Flujo de secuencia*, *Flujo de mensaje* y *Asociación*. En la Tabla A.2 se describen los principales objetos de conexión de BPMN.

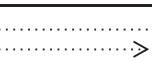
Elemento	Descripción	Notación
Flujo de secuencia	Establecen el orden de ejecución de las actividades que se realizan como parte del BP.	
Flujo de mensaje	Indican la comunicación entre dos participantes bien diferenciados (o entidades de negocio o roles del negocio) que intervienen en el BP.	
Asociación	Asocian información con objetos de flujo. Texto y objetos gráficos que no sean de flujo pueden ser asociados con objetos de flujo.	

Tabla A.2 Objetos de conexión medulares de BPMN.

A.2.3 Swinlanes

Categorizan visualmente las diferentes responsabilidades dentro de un BP. Para diferenciar los negocios y los diferentes roles, usuarios o sistemas, BPMN usa dos tipos de swimlanes: *Contenedor* y *Carril*, los cuales son descritos en la Tabla A.3.

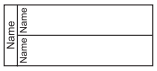
Elemento	Descripción	Notación
Contenedor	Identifica un *participante* dentro del flujo de trabajo, que es diferenciado de otro contenedor porque las actividades que realiza son diferentes a las actividades de otros contenedores.	
Carril	Es una partición dentro de una contenedor e indica quién realiza qué dentro de la empresa y dónde ocurren las actividades que contiene, con el fin de dar una mejor visión del proceso.	

Tabla A.3 Swinlanes de BPMN.

A.2.4 Artefactos

Permiten proveer información adicional acerca del BP. Aunque hay tres artefactos estandarizados en BPMN, los encargados del modelado o las herramientas de modelado pueden agregar tantos artefactos como requieran. Los principales artefactos son: *Objeto de datos*, *Grupo*, y *Anotación textual*. Éstos son descritos en la Tabla A.4.

Elemento	Descripción	Notación
Objeto de datos	Es el mecanismo que muestra los datos que son requeridos y/o producidos por las actividades. Ilustran las entradas y salidas de datos de las actividades en un BP. Son conectados a las actividades asociadas.	
Grupo	Es usado con fines de documentación o de análisis. No tiene efecto sobre el flujo de secuencia. Permite la categorización de diferentes objetos bajo una misma denominación.	
Anotación textual	Pemite incluir informacion adicional para el lector de un BPD.	

Tabla A.4 Artefactos pautados por BPMN.

Apéndice B
Ejemplo de ejecución del proceso para obtener un Autómata de Büchi a partir de una fórmula CCTL

En este apéndice se presenta un ejemplo de la ejecución del proceso de generación de un TBA a partir de una fórmula CCTL, descrito en la sección 3.5.1. La formula CCTL usada para mostrar la ejecución del algoritmo es $\vartheta = \mathsf{E}(\mathsf{F}_{[1,3]}\varphi)$, la cual expresa la verdad de la AP φ en el futuro, dentro del intervalo $[1,3]$.

B.1 Construcción del grafo

Para utilizar el proceso, debemos expresar la fórmula indicada previamente usando los operadores U y R. Entonces, $\vartheta = \mathsf{E}(\mathsf{F}_{[1,3]}\varphi) = \mathsf{E}(\neg\varphi\mathsf{U}_{[1,3]}\varphi)$, la cual especifica que la proposición atómica $\neg\varphi$ se mantendrá hasta (U) que la proposición atómica φ sea verdad dentro del intervalo $[1,3]$. La Fig. B.1 muestra el árbol de nodos procesado durante la ejecución del sub–algoritmo para la construcción del grafo de la fórmula CCTL $\mathsf{E}(\neg\varphi\mathsf{U}_{[1,3]}\varphi)$. El nodo obtenido por la aplicación de la función $\mathsf{create\_graph}(\vartheta)$ forma la raíz del árbol; la función es descrita en las líneas 7–12 de la Fig. 3.5 (sección 3.5.1), la cual contiene a la fórmula ϑ en el campo *Unprocessed* que constituye la propiedad a la que le buscamos el TBA.

La interpretación de las líneas que conectan dos nodos en el árbol de la Fig. B.1 es la siguiente: líneas punteadas significan que un nodo (el superior) ha sido dividido en dos nodos (los inferiores), mientras que las flechas de línea continua representan que el nodo inferior es sucesor del nodo superior (note que en estos casos el nodo inferior tiene en el campo *Predecessor* el identificador del nodo superior y el mismo conjunto de fórmulas que el nodo superior tiene en el campo *Next*).

Cuando un nodo es dividido en dos, el nodo colocado a la derecha de los nodos derivados representa la actualización del nodo superior (note que ambos nodos tienen el mismo nombre). Por ello es que una línea punteada es dibujada para resaltar que en ese momento no está representado un nuevo nodo. Las dos ocasiones en las cuales el nodo en la Fig. B.1 es dividido, coincide con el análisis de la fórmula $\neg\varphi\mathsf{U}_{[a,b]}\varphi$, por consiguiente, la regla de expansión aplicada es la que se muestra en las líneas 20–29 de la Fig. 3.6 (sección 3.5.1), que corresponde a la aplicación de la regla de la fila 1 de la

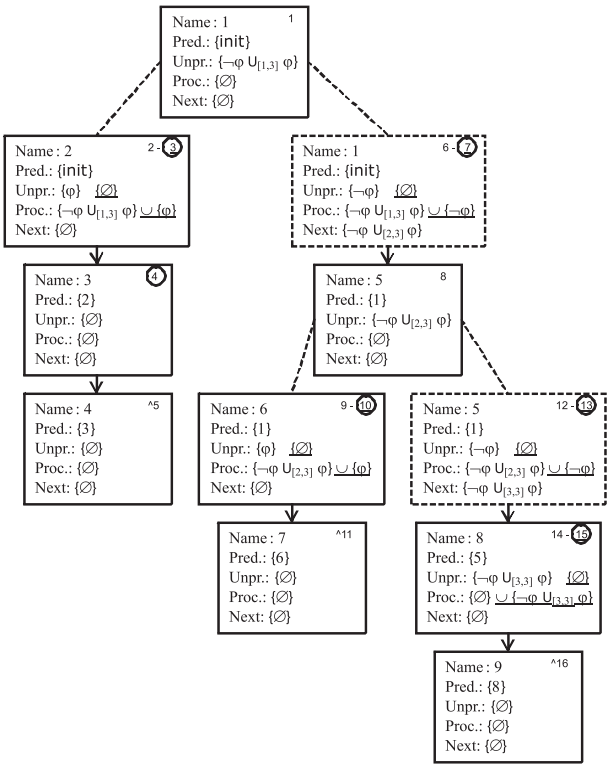


Fig. B.1 Árbol de nodos obtenido para la construcción del grafo de la fórmula $\vartheta = \mathsf{E}(\mathsf{F}_{[1,3]}\varphi)$.

Tabla 3.4 (sección 3.5.1). Entonces, al campo *Unprocessed* del nuevo nodo creado (colocado abajo a la izquierda es expandido con un solo reductor de ϑ; es decir, φ, mientras que el otro reductor (en este caso, sólo $\neg\varphi$) es agregado al campo *Unprocessed* del nodo que representa la actualización del nodo superior (colocado abajo a la derecha), y el nombre ϑ es agregado a su campo *Next*.

Los números que aparecen en la esquina superior derecha de cada nodo en la Fig. B.1 indican el orden de expansión de cada nodo. Dos números en el mismo nodo significa que el nodo fue expandido dos veces, ya que la primera vez que la función expand() (líneas 1–35 de la Fig.3.6, sección 3.5.1) fue ejecutada, se llevó a cabo una actualización del nodo, subrayando los cambios hechos, de tal manera que la nueva llamada a esta función expanda el nodo resultante de la actualización anterior (por lo tanto, el segundo de sus números también se subraya).

Observando estos números podemos constatar que es usada la estrategia DFS. Así, el nodo 1 es dividido en dos nodos (el nodo 2 es su actualización),

procesando primero el nodo 2 y todos sus sucesores antes de procesar el nodo 1 actualizado y todos sus sucesores. En la Fig. B.1, cada uno de los círculos que contienen números que indican el orden de expansión de un nodo, significan que en ese punto de la ejecución se agregó un nuevo nodo (el cual ha sido procesado) a la estructura de datos `Node_Set` (líneas 7–10 en la Fig. 3.6).

Si uno de esos números aparece precedido por el símbolo $\wedge$, significa que en ese momento de la ejecución del algoritmo el nodo ya era parte de `Node_Set` (había sido guardado en un paso previo), el cual actualiza el campo *Predecessor* (línea 8 en la Fig. 3.6). Esto es equivalente a agregarle un arco de entrada. Puede verse que cuando finaliza la función expand(), la estructura de datos `Node_Set` de ϑ tiene el contenido mostrado en la Tabla B.1 la cual corresponde con la representación gráfica de la Fig. B.2.

Name	Predecessor	Unprocessed	Processed	Next	Accept
2	{init}	$\{\varnothing\}$	$\{\neg\varphi \mathsf{U}_{[1,3]}\varphi, \varphi\}$	$\{\varnothing\}$	1
3	$\{2,3,6,8\}$	$\{\varnothing\}$	$\{\varnothing\}$	$\{\varnothing\}$	1
1	{init}	$\{\varnothing\}$	$\{\neg\varphi \mathsf{U}_{[1,3]}\varphi, \neg\varphi\}$	$\{\neg\varphi \mathsf{U}_{[2,3]}\varphi, \varphi\}$	0
6	$\{1\}$	$\{\varnothing\}$	$\{\neg\varphi \mathsf{U}_{[2,3]}\varphi, \varphi\}$	$\{\varnothing\}$	1
5	$\{1\}$	$\{\varnothing\}$	$\{\neg\varphi \mathsf{U}_{[2,3]}\varphi, \neg\varphi\}$	$\{\neg\varphi \mathsf{U}_{[3,3]}\varphi, \varphi\}$	0
8	$\{5\}$	$\{\varnothing\}$	$\{\neg\varphi \mathsf{U}_{[3,3]}\varphi, \varphi\}$	$\{\varnothing\}$	1

Tabla B.1 `Node_Set` de la fórmula $\vartheta = \mathsf{E}(\mathsf{F}_{[1,3]}\varphi)$.

Note que en la estructura de datos `Node_Set` (ver Tabla B.1) que el campo *Unprocessed* de cada nodo está vacío ($\{\varnothing\}$), lo cual significa que la ejecución del algoritmo sólo retorna nodos completamente expandidos. El grafo mostrado en la Fig. B.2 es obtenido a partir de la Tabla B.1. Las transiciones son dadas por el campo *Predecessor* de cada nodo, de tal manera que para cada $n \in Predecessor(m)$ es dibujada una transición del tipo $n \rightarrow m$. Si $\mathsf{init} \in Predecessor(m)$, entonces m es un nodo inicial (el caso de los nodos 1 y 2, los cuales han sido marcados con el símbolo $>$ en la Fig. B.2). La etiqueta asociada con cada nodo corresponde al contenido almacenado en el campo *Processed*. Cuando un nodo no tiene contenido en ese campo, tal como el nodo 3, entonces se etiqueta con el valor **T** (*True*), el cual representa que cualquier combinación de literales se cumple en ese nodo.

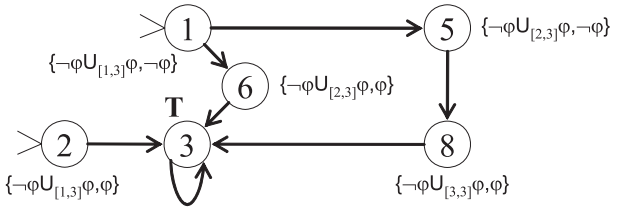


Fig. B.2 Grafo generado para la fórmula $\vartheta = \mathsf{E}(\mathsf{F}_{[1,3]}\varphi)$.

B.2 Generación del Autómata de Büchi

Ahora se muestra la ejecución del sub–algoritmo para generar el TBA a partir de la estructura de datos generada por la ejecución del sub–algoritmo de construcción del grafo. Como resultado de la ejecución del algoritmo descrito en la sección 3.5.1.4, nosotros obtenemos la Tabla B.2, la cual contiene el registro *AcceptingConds_Set* para la fórmula $\mathsf{E}(\neg\varphi\mathsf{U}_{[1,3]}\varphi)$, que permite determinar los estados de aceptación necesarios para finalizar la construcción del TBA.

Key	SubformulasSet
$\neg\varphi\mathsf{U}_{[1,3]}\varphi$	$\{\neg\varphi\mathsf{U}_{[2,3]}\varphi, \neg\varphi\mathsf{U}_{[3,3]}\varphi\}$

Tabla B.2 `AcceptanceConds_Set` de la fórmula $\mathsf{E}(\neg\varphi\mathsf{U}_{[1,3]}\varphi)$.

De acuerdo con la Tabla B.2, las sub–fórmulas $\neg\varphi\mathsf{U}_{[2,3]}\varphi$, y $\neg\varphi\mathsf{U}_{[3,3]}\varphi$ son las eventualidades que afectan la satisfacción de la fórmula $\neg\varphi\mathsf{U}_{[1,3]}\varphi$; es decir, ellas forman el conjunto de condiciones de aceptación de la fórmula. Con este resultado, el campo *Accept* de la Tabla B.1 es llenado, obteniendo que los nodos 2, 3, 6 y 8, corresponden a estados de aceptación (valor 1) para la fórmula $\mathsf{E}(\varphi\mathsf{U}_{[1,3]}\neg\varphi)$, mientras que los nodos 1 y 5, representan estados de no aceptación (valor 0). En otras palabras, los nodos 2, 3, 6 y 8, no tienen eventualidades a satisfacer, y los nodos 1 y 5, tienen eventualidades que no son satisfechas.

La Fig. B.3, generada por el software de visualización GRAPHVIZ[1], muestra el TBA obtenido del grafo mostrado en la Fig. B.2, usando directamente la estructura de datos `Node_Set`.

Ya con esta visualización, la cual es opcional, hemos mostrado una ejecución del proceso propuesto para obtener un TBA semánticamente equivalente a una fórmula CCTL.

[1] `http://www.graphviz.org/`.

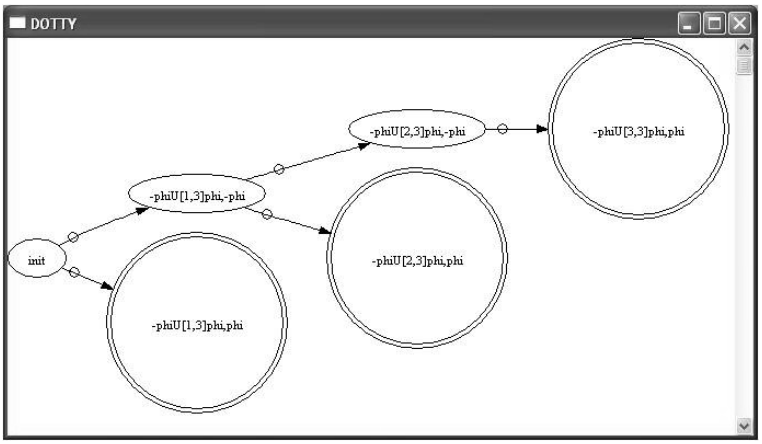


Fig. B.3 TBA de la fórmula $\mathsf{E}(\neg\varphi\mathsf{U}_{[1,3]}\varphi)$.

Apéndice C
Ejemplo de transformación de un Autómata de Büchi a términos de CSP+T

En este apéndice se presenta un ejemplo de la aplicación de las reglas para generar un término de proceso CSP+T a partir de un TBA, compiladas en la Definición 3.8.

Vamos a usar la fórmula CCTL $\vartheta = \mathsf{E}(\mathsf{F}_{[1,3]}\varphi)$, de la cual ya se obtuvo el TBA semánticamente equivalente a ella (ver Apéndice B) como resultado de la ejecución del proceso descrito en la sección 3.5.1.

En la Tabla C.1 se presenta el término de proceso CSP+T, llamado $EJEMPLO$, obtenido a partir del TBA construido en el Apéndice B.

$$
\begin{aligned}
EJEMPLO = \star.t_0 &\rightarrow P(init) \\
P(init) &= (\mathrm{I}(2,1).e_{\neg\varphi} \rightarrow P(S_1)) \,\Box\, (\mathrm{I}(2,1).e_{\varphi} \rightarrow P(S_2) \\
P(S_1) &= \ (\mathrm{I}(1,2).e_{\neg\varphi} \rightarrow P(S_3)) \,\Box\, (\mathrm{I}(1,2).e_{\varphi} \rightarrow P(S_4) \\
P(S_2) &= \ \textit{SKIP} \rightarrow EJEMPLO \\
P(S_3) &= \ \mathrm{I}(0,3).e_{\varphi} \rightarrow P(S_5) \\
P(S_4) &= \ \textit{SKIP} \rightarrow EJEMPLO \\
P(S_5) &= \ \textit{SKIP} \rightarrow EJEMPLO
\end{aligned}
$$

Tabla C.1 Término de proceso CSP+T resultante de la transformación del TBA.

Apéndice D
Aspectos prácticos del esquema conceptual de verificación composicional

En este apéndice se compilan los aspectos prácticos que fundamentan al esquema conceptual que le da soporte al EFVC presentado en el capítulo 4. Como se constata en el capítulo 4 de este tabajo, los conceptos descritos en este apéndice de manera práctica son aplicables tanto a los componentes de un sistema como los elementos que conforman un BP. La razón fundamental que justifica esto es que ambos sistemas (el de software y el de negocio) pueden ser interpretados bajo la *Teoría General de los Sistemas* [25, 104], lo que implica que *todo sistema puede ser descrito a partir de sus partes y las relaciones que existen entre ellas* [25, 104].

D.1 Generalidades

El esquema de verificación composicional se basa en el enfoque modular, el cual utiliza la descomposición y los conceptos de abstracción/refinamiento en conjunto con la técnica de MC para verificar las partes que conforman un sistema, mitigando los aspectos de complejidad [159, 19]. Cada componente del SCS es comprobado individualmente, y las propiedades locales son garantizadas por los resultados de la verificación local. Entonces, las características globales del sistema son obtebidas a partir de las propiedades verificadas localmente [19, 31, 81, 106, 159].

El concepto más importante detrás de este esquema es el razonamiento composicional, una metodología que describe un sistema sobre la base del comportamiento de sus componentes, y la forma como se componen o ensamblan [19, 159]. Este razonamiento está soportado en el paradignma Asumir/Garantizar: *cada componente garantiza ciertas propiedades sobre la base de los supuestos de los otros componentes*. Por ejemplo, dados dos componentes C_1 y C_2 y la especificación **S**, nosotros queremos saber si la composición de ambos componentes satisfacen **S**. Formalmente, si $C_1 \| C_2 \models \mathbf{S}$. Dado que razonar directamente acerca de $C_1 \| C_2$ conlleva a una explosión de estados, las técnicas de razonamiento composicional están diseñadas para razonar acerca de C_1 aisladamente de C_2, y viceversa, para obtener conclusiones sobre $C_1 \| C_2$.

Adicionalmente, es importante tener en cuenta que los aspectos que son descritos en este apéndice respetan la estructura de sincronización de los componentes del sistema [64]; es decir, que cuando se aplica la *Descomposición*, la *Abstracción*, el *Refinamiento* y el *Modelado*, se preserva en todo momento la estructura original dada por la arquitectura del sistema. En caso contrario, por el incorrecto desarrollo del proceso de *Refinamiento* no se puede asegurar que la *Abstracción* del comportamiento esperado del sistema, representado por el *Modelado* en tiempo de diseño, tenga una corresponda con el comportamiento real de sus componentes constituyentes en tiempo de ejecución, ya que no se puede garantizar que la *Descomposición* del sistema preserva la *trazabilidad* [119] que debe existir entre elementos de diseño y elementos de implementación del sistema.

Se comienza dividiendo el sistema en partes más pequeñas, los *subsistemas*. Entonces cada subsistema es especificado formalmente a un nivel adecuado de abstracción/refinamiento como un conjunto de entidades simples (componentes) y su comportamiento correcto (la especificación) es demostrado localmente. Luego, la característica global deseada del sistema es obtenida a partir de las especificaciones de todos los componentes. No se necesita más información sobre las estructuras internas de los componentes (*principio de caja negra* [89, 226]). A continuación se presentan los aspectos prácticos de los pasos del esquema conceptual presentado en la Fig. 4.1.

D.2 Descomposición

Se realiza la división inicial del sistema en entidades más pequeñas de modelado, los subsistemas. Como resultado del proceso de *refinamiento* (ver sección D.3), cada subsistema se divide en entidades más pequeñas, los componentes. Normalmente, la *descomposición* no implica trabajo de especificación formal; la estrategia de descomposición proporciona simplemente una estructura para llevar a cabo los siguientes pasos. Sin embargo, debido a la importancia de garantizar que el componente cumpla con las propiedades deseadas como parte de la definición del sistema, el lenguaje de especificación debe tener la capacidad de permitir la recomposición de los componentes especificados por separado.

Para nosotros, la mejor recomendación es descomponer los subsistemas hasta que se consigan los componentes más pequeños posible; aquellos cuyo comportamiento puede ser descrito por *una línea de control sencilla*. Así, el comportamiento puede ser especificado por un único autómata, o máquina de estado, o proceso (esto depende del formalismo usado para modelar el

comportamiento del sistema). Éstos corresponden a los procesos locales en la Fig. 4.1.

D.3 Abstracción, refinamiento y modelado

El sistema es desarrollado como resultado de un *refinamiento* gradual, partiendo de la fase de especificación de requerimientos y *añadiendo detalles progresivamente hasta alcanzar una implementación admisibl*e [9]. El esquema composicional descrito integra las vistas *estática* (estructural) y *dinámica* (comportamental) del diseño de sistemas. Por un lado, se obtiene el diseño arquitectónico del sistema, el cual representa un alto nivel de *abstracción* del mismo, su estructura. Por el otro, se obtiene una descripción completa del sistema, la cual especifica su comportamiento. Al finalizar los procesos de abstracción, refinamiento y modelado, cada componente debe ser *modelado* o mapeado a un lenguaje formal que permita su verificación. Este modelado o transformación a un lenguaje formal hace posible la definición matemática del sistema, facilitando así el análisis formal para detectar inconsistencias, ambigüedades y omisiones del comportamiento de los componentes del sistema.

D.4 Verificación local

Una vez que se obtienen la especificación y el modelo del diseño del sistema y se expresan en un dominio semántico común, el Ingeniero de Software (*Software Engineer*, SE) puede verificar si los modelos de los componentes del sistema satisfacen las especificaciones de las propiedades locales (*verificación local*), utilizando la técnica de MC junto con la herramienta de verificación formal adecuada. Gracias a la técnica de MC es posible estudiar las situaciones de comportamiento que pueden ser clasificados como *normales* o *anormales*. Si el resultado de la verificación de un componente no es satisfactoria (es decir, muestra un comportamiento anormal), la herramienta de MC generalmente presenta un contraejemplo que es usado para corregir el error en el comportamiento o en la especificación de la propiedad que se verifica. El modelo de cada componente del sistema debe ser evaluado contra su especificación formal, lo cual permite la automatización completa de este paso. Como resultado, se verifica cada componente por separado según las propiedades locales utilizando hipótesis justificadas sobre su entorno.

D.5 Verificación global

Con la finalidad de proveer la comprobación de las propiedades globales del sistema, los modelos de los procesos locales (el comportamiento de los componentes) y las especificaciones de sus propiedades (el comportamiento esperado) son compuestos usando el operador que contempla el lenguaje de especificación utilizado. La idea es entonces combinar los resultados locales obtenidos del proceso de verificación local a fin de obtener la propiedad global del sistema completo.

Formalmente, los aspectos prácticos descritos anteriormente son combinados como se explica en la sección 4.3. Como resultado, podemos obtener la completa verificación del sistema usando el Teorema de verificación composicional de sistemas (Teorema 4.1), el cual es demostrado en la sección 4.4.

Apéndice E
Anotaciones temporales en BPMN

En este apéndice se detallan las anotaciones de tiempo que deben hacerse en el modelo diseñado con en el editor de modelos BPMN para especificar los valores del tiempo de inicio y finalización de una actividad que así lo requiera. Tal como lo indica la especificación de BPMN [158], se asume que los eventos que no manejan tiempo y los flujos de secuencia no consumen un tiempo que pueda ser considerado de impacto sobre el rendimiento de un BP; por ello, éstos no son considerados dentro de nuestra propuesta (para mayores detalles, revisar el análisis hecho en la sección 3.3.2).

Las anotaciones descritas en este apéndice están basadas en el uso de los operadores de CSP+T, tal como se describió en la sección 3.4.3. Las mismas son usadas por la herramienta BTRANSFORMER (ver capítulo 5) para generar el modelo en CSP+T correspondiente al modelo fuente BPMN.

El valor de los tiempos de las anotaciones están dados en segundos, denotados a través de la tupla $\langle año, mes, día, hora, minuto, segundo\rangle$, de acuerdo a la función sec (asumimos que un mes tiene 30 días y un año 365 días) [218]:

$$sec : Time \to \mathbb{N}$$

$$sec = \lambda\ Time \bullet$$
$$año * 31556926 + mes * 2629744 + día * 86400 + hora * 3600 + minuto * 60 + segundo$$

Es posible indicar únicamente segundos, o minutos y segundos, o días, o meses, colocando el valor cero en el lugar de la tupla que se desea inhabilitar; así, las tuplas $\langle 1,0,0,0,0,0\rangle$, $\langle 0,0,2,0,0,0\rangle$ y $\langle 0,0,3,2,15,0\rangle$, denotan un (1) año, dos (2) días y tres (3) días dos (2) horas y quince (15) minutos, respectivamente. En segundos, las expresiones anteriores corresponden a 31556926 segundos, 86400 segundos y 267300 segundos, respectivamente.

En la Fig. E.1 se presenta la anotación que corresponde con los tiempos mínimo (min) y máximo (max) para la actividad/tarea S1. Cabe indicar que la validación de la relación min$\leq$max está fuera del alcance de la herramienta BTRANSFORMER, por lo que el encargado de diseñar el modelo BPMN debe velar que los valores numéricos para las variables min y max no entren en contradicción; es decir, que siempre se cumpla la relación

min≤max, ya que sino se está incurriendo en un error de modelado a nivel de BPMN que formará parte del modelo CSP+T.

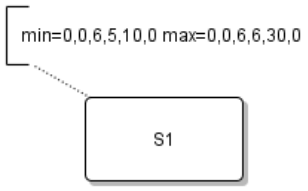


Fig. E.1 Anotación en BPMN de los tiempos min y max de una actividad/tarea.

Para hacer explícito el nombre de una variable marcadora como parte de la anotación, ésta debe indicarse sin valor asignado y agregarse al principio de la anotación, antes de los valores para las variables min y max. En la Fig. E.2, se muestra este caso, haciendo explícito el nombre de la variable marcadora v_S1 para el inicio de la actividad/tarea S1. Esta anotación es opcional, ya que la herramienta BTRANSFORMER designa automáticamente las variables marcadoras que sean necesarias para poder completar la especificación de los procesos CSP+T cada aquellas actividades/tareas que tengan valores para las variables min y max.

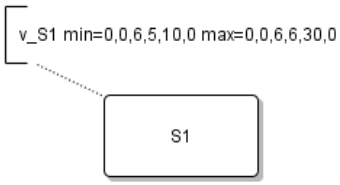


Fig. E.2 Anotación en BPMN de una variable marcadora para una actividad/tarea.

Finalmente, es posible utilizar expresiones aritméticas en una anotación para representar los valores de las variables, tal como se muestra en la Fig. E.3.

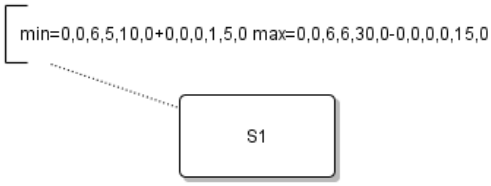


Fig. E.3 Inclusión de expresiones aritméticas en una anotación en BPMN de los tiempos min y max de una actividad/tarea.

Apéndice F
Modelo y especificación del proceso de negocio Vender Producto/Servicio

En este apéndice se detallan la sintaxis, las propiedades y los términos de proceso CSP de los participantes *Customer* y *Company* involucrados en la realización del BP VPS que utilizamos en el capítulo 6.

F.1 Sintaxis de los participantes

A continuación se presenta el esquema que describe la sintaxis del participante *Customer* y posteriormente se da la sintaxis del participante *Company*, incluyendo el sub–proceso *co_s2* que forma parte de su definición.

Escribimos $a_1 \,..\, a_n \rightsquigarrow \varnothing$ dentro de algunos esquemas para especificar que los atributos $s.a_1 \,..\, s.a_n$ de un elemento notacional de BPMN están vacíos. En particular, esto lo utilizamos en la especificación para denotar cuando un elemento notacional no tiene restricciones temporales (que llamamos *elemento de BPMN no temporizado*); es decir, colocamos $\langle\!|\ min, max\ |\!\rangle \rightsquigarrow \varnothing$ para especificar que la duración de un elemento BPMN (por ejemplo, una Activity) no tiene asignado ningún rango de duración. En contraste, para el caso de aquellos elementos notacionales cuya duración es cero, se les asigna el valor $zero_T$ (como los elementos BPMN *start*, *end*, *abort* o *agate*).

$$
\begin{array}{|l}
Customer : PLName;\ cu\_s1, cu\_s2, cu\_s3, cu\_s4, cu\_s5, cu\_s6, cu\_s7 : Task \\
\hline
\exists\, local : Local;\ t1, t2, t3, t4, t5, t6, t7, t8, t9 : Transition;\ k : \mathbb{N}; \\
\quad m1, m2, m3, m4, m5, m6, m7, m8, can : Messageflow \bullet \\
\quad states^{\sim}(local\ Customer) = \\
\qquad \{\langle\!|\ type \rightsquigarrow start.1, out \rightsquigarrow \{t1\}, loopMax \rightsquigarrow 0, \langle\!|\ min, max \rightsquigarrow zero_T\ |\!\rangle, \\
\qquad\quad break, send, receive, accept, reply, in, link, depend, error, exit \rightsquigarrow \varnothing\ |\!\rangle, \\
\qquad \langle\!|\ type \rightsquigarrow task\ cu\_s1, in \rightsquigarrow \{t1\}, out \rightsquigarrow \{t2\}, send \rightsquigarrow \{m1\}, \\
\qquad\quad accept \rightsquigarrow \{m2\}, loopMax \rightsquigarrow 0, \langle\!|\ min \rightsquigarrow zero_T, max \rightsquigarrow \infty\ |\!\rangle, \\
\qquad\quad break, receive, reply, link, depend, error, exit \rightsquigarrow \varnothing\ |\!\rangle, \\
\qquad \langle\!|\ type \rightsquigarrow miseq\ cu\_s2\ k, in \rightsquigarrow \{t2\}, out \rightsquigarrow \{t3\}, send \rightsquigarrow \{m3\}, \\
\qquad\quad accept \rightsquigarrow \{m4\}, loopMax \rightsquigarrow 0, \langle\!|\ min \rightsquigarrow zero_T, max \rightsquigarrow \infty\ |\!\rangle, \\
\qquad\quad break, receive, reply, link, depend, error, exit \rightsquigarrow \varnothing\ |\!\rangle, \\
\qquad \langle\!|\ type \rightsquigarrow xgate.1, in \rightsquigarrow \{t3\}, out \rightsquigarrow \{t4, t5\}, loopMax \rightsquigarrow 0, \langle\!|\ min, max \rightsquigarrow zero_T\ |\!\rangle, \\
\qquad\quad break, send, receive, accept, reply, link, depend, error, exit \rightsquigarrow \varnothing\ |\!\rangle,
\end{array}
$$

$$
\begin{array}{l}
\langle\!\langle\ type \rightsquigarrow task\ cu\_s3, in \rightsquigarrow \{t4\}, out \rightsquigarrow \{t5\}, loopMax \rightsquigarrow 0, \langle\!\langle\ min \rightsquigarrow zero_T, max \rightsquigarrow \infty\ \rangle\!\rangle, \\
\quad break, send, receive, accept, reply, link, depend, error, exit \rightsquigarrow \varnothing\ \rangle\!\rangle, \\
\langle\!\langle\ type \rightsquigarrow imessage\ cancel, in \rightsquigarrow \{t5\}, out \rightsquigarrow \{t6\}, send \rightsquigarrow \{can\}, \\
\quad loopMax \rightsquigarrow 0, \langle\!\langle\ min, max \rightsquigarrow zero_T\ \rangle\!\rangle, \\
\quad break, receive, accept, reply, link, depend, error, exit \rightsquigarrow \varnothing\ \rangle\!\rangle, \\
\langle\!\langle\ type \rightsquigarrow task\ cu\_s4, in \rightsquigarrow \{t7\}, out \rightsquigarrow \{t8\}, loopMax \rightsquigarrow 0, \langle\!\langle\ min \rightsquigarrow zero_T, max \rightsquigarrow \infty\ \rangle\!\rangle, \\
\quad break, send, receive, reply, accept, link, depend, error, exit \rightsquigarrow \varnothing\ \rangle\!\rangle, \\
\langle\!\langle\ type \rightsquigarrow task\ cu\_s5, in \rightsquigarrow \{t8\}, out \rightsquigarrow \{t9\}, send \rightsquigarrow \{m5\}, \\
\quad accept \rightsquigarrow \{m6\}, loopMax \rightsquigarrow 0, \langle\!\langle\ min \rightsquigarrow zero_T, max \rightsquigarrow \infty\ \rangle\!\rangle, \\
\quad break, receive, reply, link, depend, error, exit \rightsquigarrow \varnothing\ \rangle\!\rangle, \\
\langle\!\langle\ type \rightsquigarrow task\ cu\_s6, in \rightsquigarrow \{t9\}, out \rightsquigarrow \{t10\}, send \rightsquigarrow \{m8\}, \\
\quad accept \rightsquigarrow \{m7\}, loopMax \rightsquigarrow 0, \langle\!\langle\ min \rightsquigarrow zero_T, max \rightsquigarrow \infty\ \rangle\!\rangle, \\
\quad break, receive, reply, link, depend, error, exit \rightsquigarrow \varnothing\ \rangle\!\rangle, \\
\langle\!\langle\ type \rightsquigarrow end.1, in \rightsquigarrow \{t10\}, loopMax \rightsquigarrow 0, \langle\!\langle\ min, max \rightsquigarrow zero_T\ \rangle\!\rangle, \\
\quad break, send, receive, accept, reply, out, link, depend, error, exit \rightsquigarrow \varnothing\ \rangle\!\rangle, \\
\langle\!\langle\ type \rightsquigarrow abort.1, in \rightsquigarrow \{t6\}, loopMax \rightsquigarrow 0, \langle\!\langle\ min, max \rightsquigarrow zero_T\ \rangle\!\rangle, \\
\quad break, send, receive, accept, reply, out, link, depend, error, exit \rightsquigarrow \varnothing\ \rangle\!\rangle\}
\end{array}
$$

Podemos observar en el esquema de la definición sintáctica del participante *Company*, que se han colocado rangos diferentes de duración para sus tareas. De esta manera se logra precisar el tiempo de duración del flujo de trabajo que realiza el participante *Company* y ajustarlo a las reglas de negocio o niveles de QoS que establezca la organización. Aquí estamos mostrando un posible escenario de ejecución de este proceso, ya que cada variación en los tiempos de ejecución de las distintas actividades puede representar un escenario distinto del BP VPS que puede ser objeto de análisis.

Por su parte, veremos a continuación que las tareas que realiza un *Customer* dentro del BP VPS, no tienen asociado algún tiempo de duración (es decir, $\langle\!\langle\ min, max\ \rangle\!\rangle \rightsquigarrow \varnothing$), debido a que en casos reales no podemos precisar y/o controlar los tiempos de respuesta de los *Customer*. En caso de que una organización requiera fijar un tiempo de respuesta por parte del *Customer* para controlar el tiempo de duración de los BP, éstos se traducen en tiempos de espera modelados a través de flujos de excepción temporizados asociados a las actividades internas de una *Company*, y que se activan en caso de no obtener respuesta dentro de un límite de tiempo.

$$
\begin{array}{l}
Company : PLName;\ co\_s2 : BName; \\
\quad co\_s1, co\_s3, co\_s4, co\_s5, co\_s6, co\_s7, co\_s8 : Task \\
\hline
\exists local : Local;\ t1, t2, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15 : Transition; \\
\quad m1, m2, m3, m4, m5, m6, m7, m8, can : Messageflow \bullet \\
\quad states^{\sim}(local\ Company) = \\
\qquad \{\langle\!\langle\ type \rightsquigarrow start.2, out \rightsquigarrow \{t1\}, loopMax \rightsquigarrow 0, \langle\!\langle\ min, max \rightsquigarrow zero_T\ \rangle\!\rangle, \\
\qquad\quad break, send, receive, accept, reply, in, link, depend, error, exit \rightsquigarrow \varnothing\ \rangle\!\rangle,
\end{array}
$$

$$
\begin{array}{l}
\langle\, type \rightsquigarrow task\ co\_s1, in \rightsquigarrow \{t1\}, out \rightsquigarrow \{t2\}, send \rightsquigarrow \{m2\}, accept \rightsquigarrow \{m1\}, \\
\quad \langle\, min \rightsquigarrow \langle\, year, month, day, hour, second \rightsquigarrow 0, minute \rightsquigarrow 30 \,\rangle, \\
\qquad max \rightsquigarrow \langle\, year, month, day, minute, second \rightsquigarrow 0, hour \rightsquigarrow 1 \,\rangle \,\rangle, \\
\quad loopMax \rightsquigarrow 0, break, receive, reply, link, depend, error, exit \rightsquigarrow \varnothing \,\rangle, \\
\langle\, type \rightsquigarrow bpmn\ co\_s2, in \rightsquigarrow \{t2\}, out \rightsquigarrow \{t5\}, send \rightsquigarrow \{m4\}, \\
\quad accept \rightsquigarrow \{m3\}, loopMax \rightsquigarrow 0, exit \rightsquigarrow \{(k, t7)\}, \\
\quad \langle\, min \rightsquigarrow \langle\, year, month, day, minute, second \rightsquigarrow 0, hour \rightsquigarrow 18 \,\rangle, \\
\qquad max \rightsquigarrow \langle\, year, month, day, minute, second \rightsquigarrow 0, hour \rightsquigarrow 24 \,\rangle \,\rangle, \\
\quad break, receive, reply, link, depend, error \rightsquigarrow \varnothing \,\rangle, \\
\langle\, type \rightsquigarrow task\ co\_s3, in \rightsquigarrow \{t5\}, out \rightsquigarrow \{t8\}, send \rightsquigarrow \{m6\}, \\
\quad accept \rightsquigarrow \{m5\}, error \rightsquigarrow \{t6\}, break \rightsquigarrow \{can\}, loopMax \rightsquigarrow 0, \\
\quad \langle\, min \rightsquigarrow zero_T, max \rightsquigarrow \langle\, year, month, day, hour, second \rightsquigarrow 0, minute \rightsquigarrow 10 \,\rangle \,\rangle, \\
\quad receive, reply, link, depend, exit \rightsquigarrow \varnothing \,\rangle, \\
\langle\, type \rightsquigarrow task\ co\_s4, in \rightsquigarrow \{t6\}, out \rightsquigarrow \{t7\}, loopMax \rightsquigarrow 0, \\
\quad \langle\, min \rightsquigarrow \langle\, year, month, day, hour, second \rightsquigarrow 0, minute \rightsquigarrow 5 \,\rangle, \\
\qquad max \rightsquigarrow \langle\, year, month, day, hour, second \rightsquigarrow 0, minute \rightsquigarrow 10 \,\rangle \,\rangle, \\
\quad break, send, receive, accept, reply, link, depend, error, exit \rightsquigarrow \varnothing \,\rangle, \\
\langle\, type \rightsquigarrow agate.1, in \rightsquigarrow \{t8\}, out \rightsquigarrow \{t9, t11\}, loopMax \rightsquigarrow 0, \langle\, min, max \rightsquigarrow zero_T \,\rangle, \\
\quad break, send, receive, accept, reply, link, depend, error, exit \rightsquigarrow \varnothing \,\rangle, \\
\langle\, type \rightsquigarrow task\ co\_s5, in \rightsquigarrow \{t9\}, out \rightsquigarrow \{t10\}, loopMax \rightsquigarrow 0, \\
\quad \langle\, min \rightsquigarrow \langle\, year, month, hour, minute, second \rightsquigarrow 0, day \rightsquigarrow 1 \,\rangle, \\
\qquad max \rightsquigarrow \langle\, year, month, hour, minute, second \rightsquigarrow 0, day \rightsquigarrow 2 \,\rangle \,\rangle, \\
\quad break, send, receive, accept, reply, link, depend, error, exit \rightsquigarrow \varnothing \,\rangle, \\
\langle\, type \rightsquigarrow task\ co\_s6, in \rightsquigarrow \{t11\}, out \rightsquigarrow \{t12\}, loopMax \rightsquigarrow 0, \\
\quad \langle\, min \rightsquigarrow zero_T, max \rightsquigarrow \langle\, year, month, day, hour, second \rightsquigarrow 0, minute \rightsquigarrow 10 \,\rangle \,\rangle, \\
\quad break, send, receive, accept, reply, link, depend, error, exit \rightsquigarrow \varnothing \,\rangle, \\
\langle\, type \rightsquigarrow agate.2, in \rightsquigarrow \{t10, t12\}, out \rightsquigarrow \{t13\}, loopMax \rightsquigarrow 0, \langle\, min, max \rightsquigarrow zero_T \,\rangle, \\
\quad break, send, receive, accept, reply, link, depend, error, exit \rightsquigarrow \varnothing \,\rangle, \\
\langle\, type \rightsquigarrow task\ co\_s7, in \rightsquigarrow \{t13\}, out \rightsquigarrow \{t14\}, loopMax \rightsquigarrow 0, \\
\quad \langle\, min \rightsquigarrow \langle\, year, month, day, minute, second \rightsquigarrow 0, hour \rightsquigarrow 18 \,\rangle, \\
\qquad max \rightsquigarrow \langle\, year, month, day, minute, second \rightsquigarrow 0, hour \rightsquigarrow 24 \,\rangle \,\rangle, \\
\quad break, send, receive, accept, reply, link, depend, error, exit \rightsquigarrow \varnothing \,\rangle, \\
\langle\, type \rightsquigarrow task\ co\_s8, in \rightsquigarrow \{t14\}, out \rightsquigarrow \{t15\}, send \rightsquigarrow \{m9\}, \\
\quad accept \rightsquigarrow \{m10\}, loopMax \rightsquigarrow 0, \\
\quad \langle\, min \rightsquigarrow \langle\, year, month, hour, minute, second \rightsquigarrow 0, day \rightsquigarrow 1 \,\rangle, \\
\qquad max \rightsquigarrow \langle\, year, month, hour, minute, second \rightsquigarrow 0, day \rightsquigarrow 4 \,\rangle \,\rangle, \\
\quad break, receive, reply, link, depend, error, exit \rightsquigarrow \varnothing \,\rangle, \\
\langle\, type \rightsquigarrow end.2, in \rightsquigarrow \{t15\}, loopMax \rightsquigarrow 0, \langle\, min, max \rightsquigarrow zero_T \,\rangle, \\
\quad break, send, receive, accept, reply, out, link, depend, error, exit \rightsquigarrow \varnothing \,\rangle, \\
\langle\, type \rightsquigarrow abort.2, in \rightsquigarrow \{t6\}, loopMax \rightsquigarrow 0, \langle\, min, max \rightsquigarrow zero_T \,\rangle, \\
\quad break, send, receive, accept, reply, out, link, depend, error, exit \rightsquigarrow \varnothing \,\rangle\}
\end{array}
$$

A continuación se muestra la sintaxis que corresponde al sub–proceso *co_s2*. Esta especificación se hace separada de la anterior, debido a que *co_s2* no muestra a su entorno los elementos que lo conforman; es decir, el encapsula sus procesos internos. En este sentido, aquí podemos ver la jerarquía que existe entre distintos niveles de detalle de los procesos y las actividades. Esto es sumamente importante cuando se requiere verificar la sintaxis y la semántica de un BP, ya que para respetar la jerarquía entre los procesos, nosotros consideramos (y mostramos) que lo indicado es seguir un enfoque de verificación composicional.

$$
\begin{array}{|l}
co\_s2 : BName;\ co\_s21 : Task \\
\hline
\exists\, local : Local;\ t3, t4 : Transition \bullet states^{\sim}(local\ co\_s2) = \\
\quad \{\langle\!|\ type \rightsquigarrow start.3, out \rightsquigarrow \{t3\}, loopMax \rightsquigarrow 0, \langle\!|\ min, max \rightsquigarrow zero_T\ |\rangle, \\
\quad\quad break, send, receive, accept, reply, in, link, depend, error, exit \rightsquigarrow \varnothing\ |\rangle, \\
\quad \langle\!|\ type \rightsquigarrow task\ co\_s21, in \rightsquigarrow \{t3\}, out \rightsquigarrow \{t4\}, loopMax \rightsquigarrow 0, \\
\quad\quad \langle\!|\ min \rightsquigarrow zero_T, max \rightsquigarrow \langle\!|\ year, month, day, hour, second \rightsquigarrow 0, minute \rightsquigarrow 30\ |\rangle\ |\rangle, \\
\quad\quad break, send, receive, accept, reply, link, depend, error, exit \rightsquigarrow \varnothing\ |\rangle, \\
\quad \langle\!|\ type \rightsquigarrow end.3, in \rightsquigarrow \{t4\}, loopMax \rightsquigarrow 0, \langle\!|\ min, max \rightsquigarrow zero_T\ |\rangle, \\
\quad\quad break, send, receive, accept, reply, out, link, depend, error, exit \rightsquigarrow \varnothing\ |\rangle\} \\
\end{array}
$$

F.2 Fórmulas CCTL de las propiedades

A continuación se detalla el comportamiento que se espera cumplan los participantes *Customer* y *Company* en la realización del BP VPS. Este comportamiento conforma las propiedades que el BPTM derivado del BP VPS debe satisfacer, y que se utilizan en la verificación que se lleva a cabo en la sección 6.1.

Algunos ejemplos de propiedades aplicables a BP, derivadas de las reglas de negocio son: (a) el BP debe ajustarse al tiempo de ejecución requerido para cada tarea o sub–proceso, (b) la ejecución del BP debe satisfacer un orden predefinido, (c) cualquier producto/servicio solicitado, finalmente debe ser satisfecho por el BP, (d) los participantes de un BP deben trabajar (o colaborar) de forma síncrona, (e) un producto/servicio nunca podrá ser enviado por la *Company* sin una solicitud del producto/servicio, (f) un producto/servicio no será vendido si no aparece en el catálogo, (g) un producto no puede venderse si no está disponible en el stock, (h) un producto/servicio no debe ser producido por la *Company* sin una solicitud previa de creación del producto/servicio. En la Tablas F.1 y F.2 se presentan algunas de las reglas anteriores interpretadas como fórmulas CCTL.

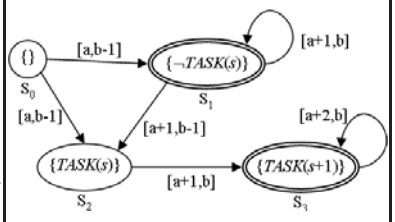
Especificación	TBA semánticamente equivalente
Propiedad: Garantía de que cada tarea cumple con el tiempo estipulado. Fórmula: $\phi_1 := \mathsf{AG}_{[a,b]}\ (TASK(s)\ \mathsf{U}_{[a+1,b]}\ TASK(s+1))$, donde s corresponde a una tarea del BP, $a := TASK(s).ran.min$ y $b := TASK(s).ran.max$. Interpretación: Cada tarea $TASK(s)$ se ejecuta dentro del intervalo de tiempo estipulado ($[TASK(s).ran.min, TASK(s).ran.max]$) y la aparición de la tarea que la precede $TASK(s+1)$ sucede dentro del intervalo de duración de la tarea que la antecede.	

Tabla F.1 Ejemplo de propiedades CCTL derivadas de las reglas de negocio para CRM – Parte 1.

Estas propiedades generales o *reglas de negocio* del BP deben ser cumplidas por el BP que se diseñe, y se presentan aquí como un subconjunto conceptualmente significante de un conjunto más amplio de propiedades específicas de cada instancia de CRM.

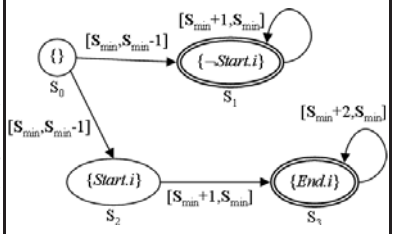
Especificación	TBA semánticamente equivalente
Propiedad: Garantía de que el BP no sobrepasa el tiempo estipulado para su ejecución. Fórmula: $\phi_2 := \mathsf{AG}_{[\mathbf{S}_{min},\mathbf{S}_{max}]}(Start.i \rightarrow \mathsf{AF}_{[\mathbf{S}_{min}+1,\mathbf{S}_{max}]}\ End.i])$, donde i representa el nivel en la jerarquía entre procesos y sub–procesos. Interpretación: Cada BP o sub–proceso respetará los tiempos estipulados como mínimo y máximo para su ejecución, por lo que el estado inicial ($Start.i$) y el estado final ($End.i$) deben ocurrir dentro del intervalo de tiempo $[\mathbf{S}_{min}, \mathbf{S}_{max}]$.	

Tabla F.2 Ejemplo de propiedades CCTL derivadas de las reglas de negocio para CRM – Parte 2.

De la propiedad ϕ_2 descrita en la Tabla F.2 podemos derivar una propiedad de vivacidad para el BPTM derivado del BP VPS, que nos permita verificar la alcanzabilidad del estado final. Entonces, dada una petición de comunicación, al final se debe recibir un recibo y una confirmación. Esta propiedad se muestra en la Tabla F.3 y es la que se utiliza en la sección 6.1, para ejemplificar la aplicación del EFVC para la verificación del BPTM derivado del BP VPS.

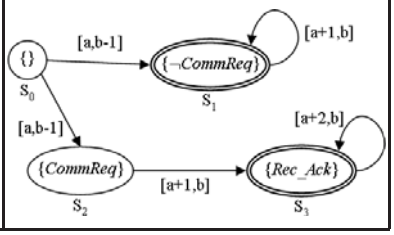
Especificación	TBA semánticamente equivalente
Propiedad: Obligación de recepción y confirmación de entrega del Producto/Servicio. Formula: $\phi_3 := \mathsf{AG}_{[a,b]}(CommReq \rightarrow \mathsf{AF}_{[a+1,b]} Rec\_Ack)$, donde $a := \mathbf{S}_{min}$ y $b := \mathbf{S}_{max}$. Interpretación: Cada petición de comunicación (*CommReq*) solicitada por el *Customer* inevitablemente en un futuro llevará a la recepción y confirmación de entrega de un Producto/Servicio (*Rec_Ack*).	

Tabla F.3 Propiedad de vivacidad del BP VPS.

F.3 Términos CSP que representan a los participantes

A continuación se presentan los términos no temporizados (especificados en CSP) que representan el comportamiento de los participantes *Customer* y *Company* que intervienen en la realización del BPTM de BP VPS. Estos términos de proceso son utilizados en la sección 6.1.5, para verificar los términos de proceso temporizados que representan a los participantes *Customer* y *Company*, utilizando el modelo no temporizado de CSP.

Utilizamos los conjuntos *CU* y *CO* definidos en la sección 6.1.3 para indexar los procesos CSP que corresponden a los elementos notaciones de los participantes *Customer* and *Company*, respectivamente. Los procesos $UT(Cus)$ y $UT(Com)$ especificados en CSP (es decir, sin algún tipo de anotación temporal) son como sigue.

$$\begin{aligned} UT(Cus) = \mathsf{let}\ X = {} & \Box i : (\alpha Y \backslash \{fin.1, abt.1\}) \bullet \\ & (i \rightarrow X \,\Box\, fin.1 \rightarrow \mathit{SKIP} \,\Box\, abt.1 \rightarrow \mathit{STOP}) \\ Y = {} & (\|i : CU \bullet \alpha\, UT(i) \circ UT(i)) \\ \mathsf{within}\ & (Y \mid [\alpha Y] \mid X) \backslash \{\mid init.Cus \mid\} \end{aligned}$$

donde por cada $i \in CU$, los procesos $UT(i)$ son definidos a continuación. Usamos $n \in \mathbb{N}$ para denotar el número de instancias de la Actividad *Product/Service information request (cu_s2)*.

$$\begin{aligned} UT(start.1) = {} & \star \rightarrow init.Cus.cu\_s1 \rightarrow \mathit{SKIP} \fatsemi fin.1 \rightarrow \mathit{SKIP} \\ UT(cu\_s1) = {} & (init.Cus.cu\_s1 \rightarrow \mathit{SKIP} \fatsemi starts.Cus.cu\_s1 \rightarrow \mathit{SKIP} \fatsemi \\ & msg.cu\_s1!x : \{in, last\} \rightarrow \mathit{SKIP} \fatsemi msg.cu\_s1.out \rightarrow \mathit{SKIP} \fatsemi \\ & init.Cus.cu\_s2 \rightarrow \mathit{SKIP} \fatsemi UT(cu\_s1)) \Box fin.1 \rightarrow \mathit{SKIP} \\ UT(cu\_s2) = {} & \\ \mathsf{let}\ A(n) = {} & n > 0\ \&\ (init.Cus.cu\_s2 \rightarrow \mathit{SKIP} \fatsemi starts.Cus.cu\_s2 \rightarrow \mathit{SKIP} \fatsemi \\ & msg.cu\_s2!x : \{in, last\} \rightarrow \mathit{SKIP} \fatsemi msg.cu\_s2.out \rightarrow \mathit{SKIP} \fatsemi \\ & init.Cus.xgate.1 \rightarrow \mathit{SKIP} \fatsemi A(n-1)) \Box init.Cus.xgate.1 \rightarrow \mathit{SKIP} \end{aligned}$$

$$
\begin{aligned}
X(n) = {} & (init.Cus.cu\_s2 \rightarrow \mathit{SKIP} \Box init.Cus.xgate.1 \rightarrow \mathit{SKIP}) \fatsemi \\
& (n > 1 \;\&\; (init.Cus.cu\_s2 \rightarrow (msg.cu\_s2.in \rightarrow X(n-1) \\
& \quad \Box msg.cu\_s2.last \rightarrow init.Cus.xgate.1 \rightarrow \mathit{SKIP})) \\
& \Box\; n = 1 \;\&\; (init.Cus.cu\_s2 \rightarrow msg.cu\_s2.last \rightarrow \\
& \quad init.Cus.xgate.1 \rightarrow \mathit{SKIP}) \\
& \Box\; n = N \;\&\; msg.cu\_s2.end \rightarrow init.Cus.xgate.1 \rightarrow \mathit{SKIP}) \\
\textsf{within}\ & ((A(n) \mid [SynSet] \mid X(n)) \fatsemi UT(cu\_s2)) \Box \mathit{fin}.1 \rightarrow \mathit{SKIP} \\
& SynSet = \{msg.cu\_s2.in, msg.cu\_s2.last, init.Cus.cu\_s2, init.Cus.xgate.1\}
\end{aligned}
$$

$$
\begin{aligned}
UT(xgate.1) = {} & (init.Cus.xgate.1 \rightarrow \mathit{SKIP} \fatsemi \\
& (init.Cus.cu\_s3 \rightarrow \mathit{SKIP} \Box init.Cus.cu\_s4 \rightarrow \mathit{SKIP}) \fatsemi \\
& UT(xgate.1)) \Box \mathit{fin}.1 \rightarrow \mathit{SKIP}
\end{aligned}
$$

$$
\begin{aligned}
UT(cu\_s3) = {} & (init.Cus.cu\_s3 \rightarrow \mathit{SKIP} \fatsemi starts.Cus.cu\_s3 \rightarrow \mathit{SKIP} \fatsemi \\
& init.Cus.cancel \rightarrow \mathit{SKIP} \fatsemi UT(cu\_s3)) \Box \mathit{fin}.1 \rightarrow \mathit{SKIP}
\end{aligned}
$$

$$
\begin{aligned}
UT(cancel) = {} & (init.Cus.cancel \rightarrow \mathit{SKIP} \fatsemi msg.cancel!x : \{can\} \rightarrow \mathit{SKIP} \fatsemi \\
& msg.cancel.out \rightarrow \mathit{SKIP} \fatsemi init.Cus.abort.1 \rightarrow \mathit{SKIP} \fatsemi \\
& UT(cancel)) \Box \mathit{fin}.1 \rightarrow \mathit{SKIP}
\end{aligned}
$$

$$
UT(abort.1) = (init.Cus.abort.1 \rightarrow \mathit{SKIP} \fatsemi abt.1 \rightarrow \mathit{STOP}) \Box \mathit{fin}.1 \rightarrow \mathit{SKIP}
$$

$$
\begin{aligned}
UT(cu\_s4) = {} & (init.Cus.cu\_s4 \rightarrow \mathit{SKIP} \fatsemi starts.Cus.cu\_s4 \rightarrow \mathit{SKIP} \fatsemi \\
& init.Cus.cu\_s5 \rightarrow \mathit{SKIP} \fatsemi UT(cu\_s4)) \Box \mathit{fin}.1 \rightarrow \mathit{SKIP}
\end{aligned}
$$

$$
\begin{aligned}
UT(cu\_s5) = {} & (init.Cus.cu\_s5 \rightarrow \mathit{SKIP} \fatsemi starts.Cus.cu\_s5 \rightarrow \mathit{SKIP} \fatsemi \\
& msg.cu\_s5!x : \{in, last\} \rightarrow \mathit{SKIP} \fatsemi msg.cu\_s5.out \rightarrow \mathit{SKIP} \fatsemi \\
& init.Cus.cu\_s6 \rightarrow \mathit{SKIP} \fatsemi UT(cu\_s5)) \Box \mathit{fin}.1 \rightarrow \mathit{SKIP}
\end{aligned}
$$

$$
\begin{aligned}
UT(cu\_s6) = {} & (init.Cus.cu\_s6 \rightarrow \mathit{SKIP} \fatsemi starts.Cus.cu\_s6 \rightarrow \mathit{SKIP} \fatsemi \\
& msg.cu\_s6!x : \{in, last\} \rightarrow \mathit{SKIP} \fatsemi msg.cu\_s6.out \rightarrow \mathit{SKIP} \fatsemi \\
& init.Cus.end.1 \rightarrow \mathit{SKIP} \fatsemi UT(cu\_s6)) \Box \mathit{fin}.1 \rightarrow \mathit{SKIP}
\end{aligned}
$$

$$
UT(end.1) = init.Cus.end.1 \rightarrow \mathit{SKIP} \fatsemi \mathit{fin}.1 \rightarrow \mathit{SKIP}
$$

El proceso $UT(Com)$ se define a continuación.

$$
\begin{aligned}
UT(Com) = {} & \textsf{let}\ Z = \Box j : (\alpha R \backslash \{\mathit{fin}.2, \mathit{abt}.2\}) \bullet \\
& \quad (j \rightarrow Z \Box \mathit{fin}.2 \rightarrow \mathit{SKIP} \Box \mathit{abt}.2 \rightarrow \mathit{STOP}) \\
& R = (\|j : CO \bullet \alpha UT(j) \circ UT(j)) \\
& \textsf{within}\ (R \mid [\alpha R] \mid Z) \backslash \{|init.Com|\}
\end{aligned}
$$

donde para cada $j \in CO$, los procesos $UT(j)$ son definidos a continuación.

$$
UT(start.2) = \star \rightarrow init.Com.co\_s1 \rightarrow \mathit{SKIP} \fatsemi \mathit{fin}.2 \rightarrow \mathit{SKIP}
$$

$$
\begin{aligned}
UT(co\_s1) = {} & (init.Com.co\_s1 \rightarrow \mathit{SKIP} \fatsemi starts.Com.co\_s1 \rightarrow \mathit{SKIP} \fatsemi \\
& msg.co\_s1!x : \{in, last\} \rightarrow \mathit{SKIP} \fatsemi msg.co\_s1.out \rightarrow \mathit{SKIP} \fatsemi \\
& init.Com.co\_s2 \rightarrow \mathit{SKIP} \fatsemi UT(co\_s1)) \Box \mathit{fin}.2 \rightarrow \mathit{SKIP}
\end{aligned}
$$

$$
\begin{aligned}
UT(co\_s2) = {} & (init.Com.co\_s2 \rightarrow \mathit{SKIP} \fatsemi msg.co\_s2!x : \{in, last\} \rightarrow \mathit{SKIP} \fatsemi \\
& msg.co\_s2.out \rightarrow \mathit{SKIP} \fatsemi starts.Com.co\_s2 \rightarrow \mathit{SKIP} \fatsemi \\
& (co\_s21 \mid [\{end.3\}] \mid end.3 \rightarrow init.Com.co\_s3 \rightarrow \mathit{SKIP}) \\
& \mid [\{init.Com.co\_s3\}] \mid \\
& init.Com.co\_s3 \rightarrow \mathit{SKIP}) \fatsemi UT(co\_s2)) \\
& \Box \mathit{fin}.2 \rightarrow \mathit{SKIP}
\end{aligned}
$$

$$
\begin{aligned}
UT(co\_s3) = {} & (init.Com.co\_s3 \rightarrow \mathit{SKIP} \fatsemi starts.Com.co\_s3 \rightarrow \\
& (\mathit{SKIP} \triangle (msg.co\_s3?x : \{cancel\} \rightarrow \mathit{SKIP} \fatsemi init.Com.co\_s4 \rightarrow \mathit{SKIP}) \Box \\
& (msg.co\_s3!x : \{in, last\} \rightarrow \mathit{SKIP} \fatsemi msg.co\_s3.out \rightarrow \mathit{SKIP} \fatsemi \\
& init.Com.agate.1 \rightarrow \mathit{SKIP}) \fatsemi UT(co\_s3))) \Box fin.2 \rightarrow \mathit{SKIP}
\end{aligned}
$$

$$
\begin{aligned}
UT(co\_s4) = {} & (init.Com.co\_s4 \rightarrow \mathit{SKIP} \fatsemi starts.Com.co\_s4 \rightarrow \mathit{SKIP} \fatsemi \\
& init.Com.abort.2 \rightarrow \mathit{SKIP} \fatsemi UT(co\_s4)) \Box fin.2 \rightarrow \mathit{SKIP}
\end{aligned}
$$

$$
UT(abort.2) = (init.Cus.abort.2 \rightarrow \mathit{SKIP} \fatsemi abt.2 \rightarrow \mathit{STOP}) \Box fin.2 \rightarrow \mathit{SKIP}
$$

$$
\begin{aligned}
UT(agate.1) = {} & (init.Com.agate.1 \rightarrow \mathit{SKIP} \fatsemi \\
& (init.Com.co\_s5 \rightarrow \mathit{SKIP} \,||\, init.Com.co\_s6 \rightarrow \mathit{SKIP} \fatsemi \\
& UT(agate.1)) \Box fin.2 \rightarrow \mathit{SKIP}
\end{aligned}
$$

$$
\begin{aligned}
UT(co\_s5) = {} & (init.Com.co\_s5 \rightarrow \mathit{SKIP} \fatsemi starts.Com.co\_s5 \rightarrow \mathit{SKIP} \fatsemi \\
& init.Com.agate.2 \rightarrow \mathit{SKIP} \fatsemi UT(co\_s5)) \Box fin.2 \rightarrow \mathit{SKIP}
\end{aligned}
$$

$$
\begin{aligned}
UT(co\_s6) = {} & (init.Com.co\_s6 \rightarrow \mathit{SKIP} \fatsemi starts.Com.co\_s6 \rightarrow \mathit{SKIP} \fatsemi \\
& init.Com.agate.2 \rightarrow \mathit{SKIP} \fatsemi UT(co\_s6)) \Box fin.2 \rightarrow \mathit{SKIP}
\end{aligned}
$$

$$
\begin{aligned}
UT(agate.2) = {} & ((UT(co\_s5) \mid [init.Com.agate.2] \mid UT(co\_s6)) \rightarrow \mathit{SKIP} \fatsemi \\
& UT(agate.2)) \Box fin.2 \rightarrow \mathit{SKIP}
\end{aligned}
$$

$$
\begin{aligned}
UT(co\_s7) = {} & (init.Com.co\_s7 \rightarrow \mathit{SKIP} \fatsemi starts.Com.co\_s7 \rightarrow \mathit{SKIP} \fatsemi \\
& init.Com.co\_s8 \rightarrow \mathit{SKIP} \fatsemi UT(co\_s7)) \Box fin.2 \rightarrow \mathit{SKIP}
\end{aligned}
$$

$$
\begin{aligned}
UT(co\_s8) = {} & (init.Com.co\_s8 \rightarrow \mathit{SKIP} \fatsemi starts.Com.co\_s8 \rightarrow \mathit{SKIP} \fatsemi \\
& msg.co\_s8!x : \{in, last\} \rightarrow \mathit{SKIP} \fatsemi msg.co\_s8.out \rightarrow \mathit{SKIP} \fatsemi \\
& init.Com.end.2 \rightarrow \mathit{SKIP} \fatsemi UT(co\_s8)) \Box fin.2 \rightarrow \mathit{SKIP}
\end{aligned}
$$

$$
UT(end.2) = init.Com.end.2 \rightarrow \mathit{SKIP} \fatsemi fin.2 \rightarrow \mathit{SKIP}
$$

El proceso $UT(co\_s2)$ queda como sigue. Utilizamos el conjunto $CO2$ de la sección 6.1.3 para indexar los procesos CSP correspondientes:

$$
\begin{aligned}
UT(co\_s2) = {} & \textsf{let } T = \Box k : (\alpha W \backslash \{fin.3\}) \bullet (k \rightarrow T \Box fin.3 \rightarrow \mathit{SKIP}) \\
W = {} & (\| k : CO2 \bullet \alpha UT(k) \circ UT(k)) \\
& \textsf{within } (W \mid [\alpha W] \mid T) \backslash \{| init |\}
\end{aligned}
$$

donde por cada $k \in CO2$, los procesos $UT(k)$ son definidos a continuación.

$$
UT(start.3) = init.Com.co\_s21 \rightarrow \mathit{SKIP} \fatsemi fin.3 \rightarrow \mathit{SKIP}
$$

$$
\begin{aligned}
UT(co\_s21) = {} & (init.Com.co\_s21 \rightarrow \mathit{SKIP} \fatsemi starts.Com.co\_s21 \rightarrow \mathit{SKIP} \fatsemi \\
& init.Com.end.3 \rightarrow \mathit{SKIP} \fatsemi UT(co\_s21)) \Box fin.3 \rightarrow \mathit{SKIP}
\end{aligned}
$$

$$
UT(end.3) = init.Com.end.3 \rightarrow \mathit{SKIP} \fatsemi fin.3 \rightarrow \mathit{SKIP}
$$

Apéndice G
Fórmulas CCTL de las propiedades para el procotolo DDBM

En este apéndice se detalla el comportamiento esperado para los componentes que realizan el protocolo DDBM descrito en la sección 6.2.

En las Tablas G.1 y G.2 se presentan las fórmulas CCTL que expresan las propiedades de *safety*, y en la Tabla G.3 las que corresponden a las propiedades de *liveness*, *deadlock–freeness* y *fairness*, que se refieren específicamente a las siguientes propiedades del modelo del sistema DDBM: (a) integridad de datos entre las diferentes DB locales del DDBM, (b) caracterización y determinación de los estados dinámicos de los mensajes transmitidos, (c) preservación del orden de recepción de mensajes, y (d) preservación de los estados de control interno en cada DDBM.

A fin de preservar la exactitud del protocolo y la integridad de los datos replicados, la propiedad ϕ_2 asegura que cualquier DDBM estará en el estado *Active* cuando el mensaje correspondiente se envía o la confirmación no se ha recibido aún. La propiedad ϕ_3 garantiza específicamente que todos los *acknowledgement messages* provenientes de los DDBMs remotos son recibidos antes de la actualización de los datos locales.

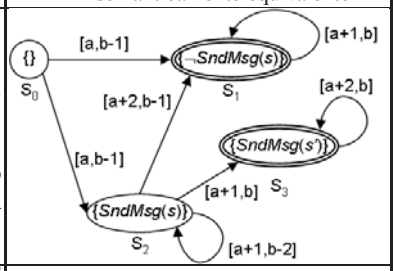
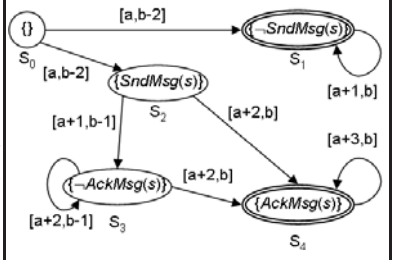
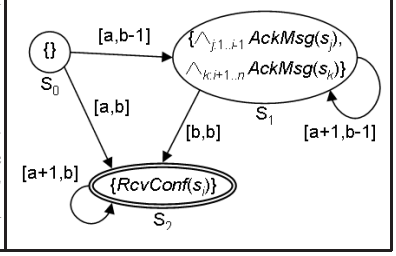
Especificación	TBA semánticamente equivalente
Propiedad: Garantía de procesar un mensaje a la vez. Fórmula: $\phi_1 := \mathsf{AG}_{[a,b]}(SndMsg(s) \rightarrow \mathsf{A}[SndMsg(s)\ \mathsf{U}_{[a+1,b-1]}\ (\neg SndMsg(s) \wedge \mathsf{A}[\neg SndMsg(s)\ \mathsf{U}_{[a+2,b]}\ SndMsg(s')])])$ Interpretación: El DDBM recibe el mensaje (s) sólo una vez antes de que el siguiente mensaje (s') sea enviado, lo cual ocurre dentro del intervalo $[a, b]$.	
Propiedad: Obligación de mantener el estado "sent" antes de que sea recibida la confirmación del mensaje. Fórmula: $\phi_2 := \mathsf{AG}_{[a,b]}(SndMsg(s) \rightarrow \mathsf{AF}_{[a+1,b-1]}[SndMsg(s)\ \mathsf{U}_{[a+1,b]}\ AckMsg(s)])$ Interpretación: Cada mensaje de datos que es generado por el DDBM iniciador es eventualmente recibido y el estado *"sent"* se mantiene hasta que el mensaje de confirmación es recibido, entonces el estado del mensaje cambia a *"acknowledged"* dentro del intervalo $[a, b]$.	
Propiedad: Garantía de la completa replicación global de datos. Fórmula: $\phi_3 = \neg(\bigwedge_{j:1..i-1} AckMsg(s_j) \bigwedge_{k:i+1..n} AckMsg(s_k))\ \mathsf{U}_{[a,b]}\ RcvConf(s_i)$ Interpretación: El DDBM iniciador de una actualización de datos en los servidores remotos debe asegurarse que todos los *acknowledgement messages* son recibidos antes de retornar a su estado inicial dentro del intervalo $[a, b]$.	

Tabla G.1 Propiedades de *safety* para el DDBM – parte 1.

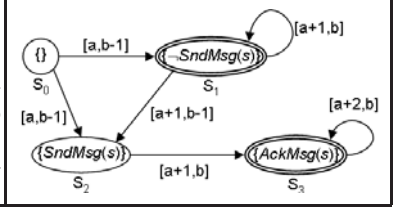
Especificación	TBA semánticamente equivalente
Propiedad: Garantía del mensaje de confirmación. Fórmula: $\phi_4 := \mathsf{AG}_{[a,b]}(SndMsg(s) \rightarrow \mathsf{AF}_{[a+1,b]} AckMsg(s))$ Interpretación: Cada mensaje de datos que es generado por el DDBM iniciador es siempre confirmado por los DDBM receptores; es decir, el estado *"sent"* de cualquier mensaje eventualmente cambiará a *"acknowledged"* dentro del intervalo $[a, b]$.	

Tabla G.2 Propiedades de *safety* para el DDBM – parte 2.

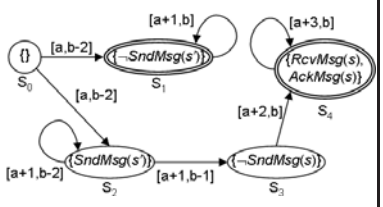
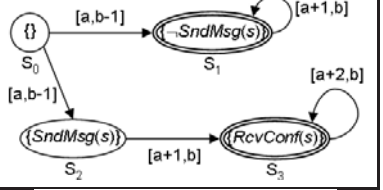
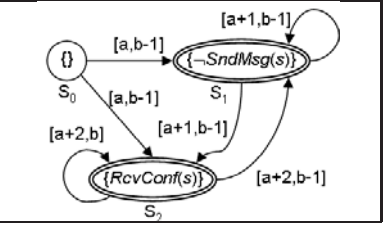
Especificación	TBA semánticamente equivalente
Propiedad: Garantía de que no se perderán mensajes mientras se están transmitiendo sobre la red. Fórmula: $\phi_5 := \mathsf{AG}_{[a,b]}(SndMsg(s') \rightarrow \mathsf{A}[SndMsg(s') \mathsf{U}_{[a+1,b-1]}(\neg SndMsg(s) \wedge \mathsf{A}[\neg SndMsg(s) \mathsf{U}_{[a+2,b]} RcvMsg(s) \wedge AckMsg(s)])])$ Interpretación: El DDBM se encuentra en el estado *sending message* (s') hasta que el receptor envía la confirmación del mensaje (s) recibido previamente, dentro del intervalo $[a, b]$.	
Propiedad: Obligación de la confirmación de mensajes. Fórmula: $\phi_6 := \mathsf{AG}_{[a,b]}[\mathsf{AF}_{[a,b-1]} SndMsg(s) \rightarrow \mathsf{AF}_{[a+1,b]} RcvConf(s)]$ Interpretación: Cada mensaje de datos que es generado por el DDBM iniciador será finalmente *confirmed* dentro del intervalo $[a, b]$.	
Propiedad: Garantía de que cada mensaje será infinitamente recibido. Fórmula: $\phi_7 := \mathsf{AG}_{[a,b]}[\mathsf{AF}_{[a,b-1]}(\neg SndMsg(s)) \vee \mathsf{AF}_{[a+1,b]} RcvConf(s)]$ Interpretación: Cada mensaje de datos que es generado por el DDBM iniciador será infinitamente ofrecido a los DDBMs receptores dentro del intervalo $[a, b]$.	

Tabla G.3 Propiedades de *deadlock–freeness*, *liveness* y *fairness*, para el DDBM.

Referencias

1. van der Aalst, W.: Verification of Workflow Nets, LNCS 1248: Proc. Application and Theory of Petri Nets 1997, pp. 407–426. Springer–Verlag, Berlin (1997)
2. van der Aalst, W.: Formalization and verification of event-driven process chains. Inf. Softw. Technol. **41**(10), 639–650 (1999)
3. van der Aalst, W.: Making Work Flow: On the Application of Petri Nets to Business Process Management, LNCS 2360: Application and Theory of Petri Nets 2002, pp. 1–22. Springer–Verlag, Berlin (2002)
4. Aalst, W.: Challenges in business process analysis. In: Enterprise Information Systems, *Lecture Notes in Business Information Processing*, vol. 12, pp. 27–42. Springer Berlin Heidelberg (2009)
5. van der Aalst, W., Hofstede, A.: Verification of workflow task structures: A petri-net-based approach. Inf. Syst. **25**(1) (2000)
6. van der Aalst, W., ter Hofstede, A., Kiepuszewski, B., Barros, A.: Workflow patterns. Distributed and Parallel Databases **14**(3)
7. Aceto, L., Ingólfsdóttir, A., Larsen, K., Srba, J.: Reactive Systems: Modelling, Specification and Verification (2007)
8. Aguilar-Savén, R.: Business process modelling: Review and framework. International Journal of Production Economics **90**(2), 129–149 (2004)
9. Allen, R., Garlan, D.: A formal basis for architectural connection. ACM Trans. Softw. Eng. Methodol. **6**(3) (1997)
10. Alpuente, M., Gallardo, M., Pimentel, E., Villanueva, A.: A semantic framework for the abstract model checking of tccp programs. Theor. Comput. Sci. **346**(1) (2005)
11. Alur, R., Dill, D.: A theory of timed automata. Theoretical Computer Science **126**(2), 183–235 (1994)
12. Aronson, J.: Mental models and deduction. American Behavioral Scientist **40**(6), 782–797 (1997)
13. Baacke, L., Mettler, T., Rohner, P.: Component-based process modelling in health care. In: Proc. 17th European Conference on Information Systems (ECIS 2009)
14. Baeten, J.: A brief history of process algebra. Theor. Comput. Sci. **335**(2-3) (2005)
15. Baier, C., Groser, M., Ciesinski, F.: Partial order reduction for probabilistic systems. In: QEST '04: Proceedings of the The Quantitative Evaluation of Systems, First International Conference (2004)
16. Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press, Cambridge, USA (2008)
17. Ball, T., Podelski, A., Rajamani, S.: Relative completeness of abstraction refinement for software model checking. In: TACAS '02: Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (2002)
18. Ball, T., Rajamani, S.: The slam project: Debugging system software via static analysis. In: POPL '02: Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (2002)
19. Barbier, F.: Composability for software components: An approach based on the whole–part theory. In: Proceedings of the Eighth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2002) (2002)
20. von der Beeck, M.: Behaviour specifications: Equivalence and refinement notions. In: Visuelle Verhaltensmodellierung Verteilter und Nebenläufiger Software–Systeme, 8. Workshop des Arbeitskreises GROOM der GI Fachgruppe 2.1.9 OO Software-Entwicklung, pp. 1–5. Universität Münster, Paderborn, GR (2000)

21. Benghazi, K., Capel, M., Holgado, J., Mendoza, L.E.: A methodological approach to the formal specification of real–time systems by transformation of UML–RT design models. Science of Computer Programming **65**(1), 41–56 (2007). Special Issue on: Increasing Adequacy and Reliability of EIS
22. Bérard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., Schnoebelen, P., McKenzie, P.: Systems and software verification: model-checking techniques and tools (1999)
23. Bergstra, J., Klop, J.: Process algebra for synchronous communication. Information and Control **60**(1–3), 109–137 (1984)
24. Bergstra, J., Klop, J.: Algebra of communicating processes with abstraction. Theoretical Computer Science **37**, 77–121 (1985)
25. Bertalanffy, L.: General System Theory: Foundations, Development, Applications
26. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic Model Checking without BDDs, LNCS 1579: Tools and Algorithms for the Construction and Analysis of Systems, pp. 193–207. Springer Berlin, Heidelberg, Germany (1999)
27. Bošnački, D.: A nested depth first search algorithm for model checking with symmetry reduction. In: FORTE '02: Proceedings of the 22nd IFIP WG 6.1 International Conference Houston on Formal Techniques for Networked and Distributed Systems (2002)
28. Bryant, R.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Comput. **35**(8) (1986)
29. Bryant, R.: Symbolic boolean manipulation with ordered binary-decision diagrams. ACM Comput. Surv. **24**(3) (1992)
30. Bull, C.: Strategic issues in customet relationship management (crm) implementation. Business Process Management Journal **9**(5), 592–602 (2003)
31. Bultan, T., Fischer, J., Gerber, R.: Compositional verification by model checking for counter–examples. In: ISSTA '96: Proc. of the 1996 ACM SIGSOFT International Symposium on Software Testing and Analysis (1996)
32. Burch, J., Clarke, E., Long, D.: Symbolic model checking with partitioned transition relations. In: VLSI 91: Proc. IFIP TC10/WG 10.5 International Conference on Very Large Scale Integration (1991)
33. Burch, J., Clarke, E., Long, D., McMillan, K., Dill, D.: Symbolic model checking for sequential circuit verification. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **13**(4), 401–424 (1994)
34. Burch, J., Clarke, E., McMillan, K., Dill, D., Hwang, L.: Symbolic model checking: 10^20 states and beyond. Inf. Comput. **98**(2), 142–170 (1992). DOI http://dx.doi.org/10.1016/0890-5401(92)90017-A
35. Burns, A., Wellings, A.: Real–time Systems and Programming Languages, third edn. Pearson Education Limited, United Kingdom (2003)
36. Capel, M., Benghazi, K., Holgado, J., Mendoza, L.: An interpretation of behavioural consistency of UML–RT diagram in terms of CSP+T. In: Proc. 5th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems (MSVVEIS 2007), pp. 74–83. INSTICC Press, Setúbal, Portugal (2007)
37. Capel, M., Mendoza, L.: Automatic Compositional Verification of Business Processes, LNBIP 24: Enterprise Information Systems, pp. 479–490. Springer Berlin, Heidelberg, Germany (2009)
38. Capel, M., Mendoza, L., Benghazi, K.: Automatic verification of business process integrity. Int. J. Simulation and Process Modelling **4**(3/4), 167–182 (2008)
39. Capel, M., Mendoza, L., Benghazi, K., Holgado, J.: A semantic formalization of UML–RT models with CSP+T processes applicable to real–time systems verification. Actas XV Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2006) **1**, 283–292 (2006)
40. Chen, I., Popovich, K.: Understanding customer relationship management (CRM). people, process, and technology. Business Process Management Journal **9**(5), 672–688 (2003)
41. Cheng, A.: Real–Time Systems: Scheduling, Analysis, and Verification. John Wiley & Sons, Inc., Hoboken, USA (2002)

42. Cheung, S., Kramer, J.: Enhancing compositional reachability analysis with context constraints. In: SIGSOFT '93: Proc. of the 1st ACM SIGSOFT Symposium on Foundations of Software Engineering (1993)
43. Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M.: Nusmv: A new symbolic model checker. International Journal on Software Tools for Technology Transfer **2**(4), 410–425 (2000)
44. Clarke, E., Biere, A., Raimi, R., Zhu, Y.: Bounded model checking using satisfiability solving. Form. Methods Syst. Des. **19**(1) (2001)
45. Clarke, E., Emerson, E.: Design and synthesis of synchronization skeletons using branching-time temporal logic. In: Logic of Programs, Workshop (1982)
46. Clarke, E., Emerson, E., Sistla, A.: Automatic verification of finite–state concurrent systems. In: ACM Symposioum on Principles of Programming Languages (1983)
47. Clarke, E., Emerson, E., Sistla, A.: Automatic verification of finite–state concurrent systems using temporal logic specifications. ACM Trans. Program. Lang. Syst. **8**(2) (1986)
48. Clarke, E., Grumberg, O., Long, D.: Model checking and abstraction. ACM Trans. Program. Lang. Syst. **16**(5) (1994)
49. Clarke, E., Grumberg, O., Peled, D.: Model Checking. MIT. The MIT Press, Cambridge, USA (2000)
50. Clarke, E., Long, D., McMillan, K.: Compositional model checking. In: Proc. of the Fourth Annual Symposium on Logic in Computer Science (1989)
51. Clarke, E., Wing, J.: Formal methods: State of the art and future directions. ACM Comput. Surv. **28**(4) (1996)
52. Cobleigh, J., Giannakopoulou, D., Păsăreanu, C.: Learning Assumptions for Compositional Verification, LNCS 2619: 9th International Conference Tools and Algorithms for the Construction and Analysis of Systems (TACAS) 2003, pp. 331–346. Springer–Verlag, Berlin (2003)
53. Combi, C., Posenato, R.: Controllability in temporal conceptual workflow schemata. In: Business Process Management, *Lecture Notes in Computer Science*, vol. 5701, pp. 64–79. Springer Berlin / Heidelberg (2009)
54. Coudert, O., Berthet, C., Madre, J.: Verification of synchronous sequential machines based on symbolic execution. In: Proceedings of the international workshop on Automatic verification methods for finite state systems (1990)
55. Cousot, P.: Program analysis: the abstract interpretation perspective. ACM Comput. Surv. p. 165 (1996). DOI http://doi.acm.org/10.1145/242224.242433
56. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: POPL '77: Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, pp. 238–252 (1977)
57. Dams, D., Gerth, R., Grumberg, O.: Abstract interpretation of reactive systems. ACM Trans. Program. Lang. Syst. **19**(2) (1997)
58. Davies, J., Schneider, S.: Using CSP to verify a timed protocol over a fair medium. In: CONCUR '92: Proceedings of the Third International Conference on Concurrency Theory (1992)
59. Davies, J., Schneider, S.: A brief history of Timed CSP. In: MFPS '92: Selected papers of the meeting on Mathematical foundations of programming semantics, pp. 243–271. Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands (1995). DOI http://dx.doi.org/10.1016/0304-3975(94)00169-J
60. Dijkman, R., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in bpmn. Inf. Softw. Technol. **50**(12) (2008)
61. Duursma, C., Olle, U.: Task model definition and task analysis process. Tech. rep., Vrije University, Brussels KADSII /M5/VUB/RR/004/2.0. (1994)
62. (ed), L.F.: 2009 BPM and Workflow Handbook: Spotlight on Government. Lighthouse Point, USA (2009)
63. Eertink, H., Janssen, W., Luttighuis, P.O., Teeuw, W., Vissers, C.: A Business Process Design Language, LNCS 1708: FM99 Formal Methods, pp. 76–95. Springer Berlin, Heidelberg, Germany (1999)

64. Emerson, E., Clarke, E.: Using branching time temporal logic to synthesize synchronization skeletons. Sci. Comput. Program. **2**(3), 241–266 (1982)
65. Emerson, E., Sistla, A.: Symmetry and model checking. Form. Methods Syst. Des. **9**(1-2) (1996)
66. Engels, G., Küster, J., Groenewegen, L.: Consistent interaction of software components. Transactions of the SDPS: Journal of Integrated Design & Process Science **6**(4), 2–22 (2003)
67. Eriksson, H.E., Penker, M.: Business Modeling With UML: Business Patterns at Work (1998)
68. Eshuis, H.: Semantics and verification of uml activity diagrams for workflow modelling. Phd thesis, University of Twente, Enschede, The Netherlands (2002)
69. Fisteus, J.A.: Definición de un modelo para la verificación formal de procesos de negocio. Phd thesis, Universidad Carlos III de Madrid, Madrid, España (2004)
70. Fisteus, J.A., Fernández, L.S., Kloos, C.D.: Formal Verification of BPEL4WS Business Collaborations, LNCS 3182: E-Commerce and Web Technologies, pp. 76–85. Springer Berlin, Heidelberg, Germany (2008)
71. Formal Systems (Europe) Ltd: Failures–Divergence Refinement – FDR2 User Manual. Formal Systems (Europe) Ltd, Oxford (2005)
72. Frehse, G., Stursberg, O., Engell, S., Huuck, R., Lukoschus, B.: Verification of hybrid controlled processing systems based on decomposition and deduction. In: ISIC'01: Proceedings of the 2001 IEEE International Symposium on Intelligent Control (2001)
73. Frehse, G., Stursberg, O., Engell, S., Huuck, R., Lukoschus, B.: Modular analysis of discrete controllers for distributed hybrid systems. In: b02: The XV IFAC World Congress, pp. 21–26. IFAC, Barcelona, Spain (2002)
74. Fuggetta, A.: A classification of case technology. Computer **26**(12), 25–38 (1993)
75. Gallardo, M., Martínez, J., Merino, P., Pimentel, E.: αspin: A tool for abstract model checking. Int. J. Softw. Tools Technol. Transf. **5**(2) (2004)
76. Gerth, R., Peled, D., Vardi, M., Wolper, P.: Simple on–the–fly automatic verification of linear temporal logic. In: Proceedings of the Fifteenth IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification XV (1996)
77. Giaglis, G.: A taxonomy of business process modeling and information systems modeling techniques. International Journal of Flexible Manufacturing Systems **13**(2) (2001)
78. Giese, H., Tichy, M., Burmester, S., Flake, S.: Towards the compositional verification of real–time UML designs. In: ESEC/FSE–11: Proc. 9th European Software Engineering Conference held jointly with 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering (2003)
79. Godefroid, P.:
80. Gottschalk, F., van der Aalst, W., Jansen-Vullers, M.: Merging event-driven process chains. In: OTM '08: Proceedings of the OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008. Part I on On the Move to Meaningful Internet Systems: (2008)
81. Grumberg, O., Long, D.: Model checking and modular verification. ACM Transaction on Programming Languages and Systems **16**(3), 843–871 (1994)
82. Hammer, M.: Reengineering work: Dont automate, obliterate. harvard business review. Harvard Business Review **jul/ago**, 104–112 (1990)
83. Hammer, M.: Beyond Reengineering: How the Process-Centered Organization is Changing Our Work and Our Lives, 1st ed. edn. Harper Business. HarperCollins Publishers Inc., New York, USA (1996)
84. Hammer, M., Champy, J.: Reengineering the Corporation: A Manifesto for Business Revolution. Harper Business Essentials. HarperCollins Publishers Inc., New York, USA (2003)
85. Harel, D., Lachover, H., Naamad, A., Pnueli, A., Politi, M., Sherman, R., A. Shtul-Trauring, a.: Statemate: a working environment for the development of complex reactive systems. In: ICSE '88: Proceedings of the 10th international conference on Software engineering (1988)
86. Harel, D., Naamad, A.: The statemate semantics of statecharts. Technical report, i-Logix (1995)
87. Harel, D., Pnueli, A.: On the development of reactive systems (1985)

88. Hatcliff, J., Dwyer, M., Păsăreanu, C., Robby: Foundations of the Bandera Abstraction Tools (2002)
89. Haugen, O., Husa, K., Runde, R., Stølen, K.: STAIRS towards formal design with sequence diagrams. Software & System Modelling **4**(4), 355–357 (2005)
90. Havey, M.: Essential Business Process Modeling (2005)
91. Hendriks, M., Behrmann, G., Larsen, K., Niebert, P., Vaandrager, F.: Adding Symmetry Reduction to UPPAAL (2004)
92. Hoare, C.: Communicating Sequential Processes. International Series in Computer Science. Prentice–Hall International Ltd., Hertfordshire UK (1985)
93. Holzmann, G.: The model checker spin. IEEE Trans. Softw. Eng. **23**(5) (1997)
94. Hopcroft, J., Motwani, R., Ullman, J.: Introduction to Automata Theory, Languages, and Computation (2006)
95. Hornos, M., Capel, M.: Automata generation for on–the–fly automatic verification using formulas of an interval logic. In: ACSD '01: Proceedings of the Second International Conference on Application of Concurrency to System Design (2001)
96. Hornos, M., Capel, M.: On–the–fly model checking from interval logic specifications. SIGPLAN Not. **37**(12), 108–119 (2002)
97. Huang, G.D., Yu, F.: Tctl inevitability analysis of dense-time systems: From theory to engineering. IEEE Trans. Softw. Eng. **32**(7) (2006). Member-Wang, Farn
98. Huckvale, T., Ould, M.: Process Modeling - Who, What and How: Role Activity Diagrammin, Business Process Change: Reengineering, Concepts, Methods and Technologies, pp. 330–349. IdeaGroup Publishing, Hershey, USA (1995)
99. Hull, R., Benedikt, M., Christophides, V., Su, J.: E-services: A look behind the curtain. In: PODS '03: Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pp. 1–14. ACM, New York, USA (2003)
100. Ip, C., Dill, D.: Better verification through symmetry. Form. Methods Syst. Des. **9**(1-2) (1996)
101. Jacobson, I., M.E., Jacobson, A.: The object advantage: business process reengineering with object technology (1994)
102. Janssen, W., Mateescu, R., Mauw, S.: Verifying business processes using spin. In: Proceedings of the 4th International SPIN Workshop, pp. 1–16. IFIP, Paris, Francia (1998)
103. Joël, O.: Discrete analysis of continuous behaviour in real-time concurrent systems. Ph.D. thesis, Oxford University
104. Johansen, O.: Introducción a la Teoría General de Sistemas
105. Jones, N., Nielson, F.: Handbook of Logic in Computer Science, chap. Abstract Interpretation: A Semantics-Based Tool for Program Analysis, pp. 527–636. Oxford University Press (1995)
106. Jonnson, B.: Compositional specification and verification of distributed systems. ACM TOPLAS **16**(2), 259–303 (1994)
107. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: Atl: A model transformation tool. Science of Computer Programming **72**(1–2), 31–39 (2008)
108. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., Valduriez, P.: Atl: a qvt-like transformation language. In: Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, OOPSLA '06 (2006)
109. Jouault, F., Kurtev, I.:
110. JTC 1/SC 7: Software Engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Guide to SQuaRE. ISO/IEC Standard 25000. International Organization for Standardization, Geneva, Switzerland (2005)
111. Juan, E., Tsai, J., Murata, T.: Compositional verification of concurrent systems using petri–net–based condensation rules. ACM Trans. Program. Lang. Syst. **20**(5) (1998)
112. Kesten, Y., Klein, A., Pnueli, A., Raanan, G.: A Perfecto Verification: Combining Model Checking with Deductive Analysis to Verify Real-Life Software, LNCS 1708: FM'99 World Congress on Formal Methods in the Development of Computing Systems, pp. 173–194. Springer–Verlag, Berlin (1999)

113. Kettinger, W., Teng, J., Guha, S.: The Process Reengineering Life Cycle Methodology: A case Study, Business Process Change: Reengineering, Concepts, Methods and Technologies, pp. 211–244. IdeaGroup Publishing, Hershey, USA (1995)
114. Kitchenham, B., Linkman, S., Law, D.: Desmet: a methodology for evaluating software engineering methods and tools. Computing & Control Engineering Journal **8**(3), 120–126 (1997)
115. Knapp, A., Merz, S., Rauh, C.: Model checking - timed uml state machines and collaborations. In: FTRTFT '02: Proceedings of the 7th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (2002)
116. Koch, N.: Classification of model transformation techniques used in uml-based web engineering. IET Software **1**(3), 98–111 (2007)
117. Koniewski, R., Dzielinski, A., Amborski, K.: Use of petri nets and business processes management notation in modelling and simulation of multimodal logistics chains. In: Proc. 20th European Conference on Modelling and Simulation (ECMS 2006)
118. Kropf, T., Rüf, J.: Using mtbdds for discrete timed symbolic model checking CCTL. In: Proc. 1997 European Design and Test Conference (ED&TC 1997), pp. 182–187 (1997)
119. Kruchten, P.: The 4+1 view model of architecture. IEEE Softw. **12**(6) (1995)
120. Kruchten, P.: The Rational Unified Process: An Introduction, Second Edition, 3rd edn. (2003)
121. Lamport, L.: Composition: A Way to Make Proofs Harder, LNCS 1536: Compositionality: The Significant Difference, pp. 402–423. Springer Berlin, Heidelberg, Germany (1998)
122. Laplante, P.: Real–Time Systems Design and Analysis: An Engineer's Handbook. IEEE Press, Piscataway, USA (1992)
123. Laplante, P.: Design and Application of Real–Time Systems (1997)
124. Loiseaux, C., Graf, S., Sifakis, J., Bouajjani, A., Bensalem, S.: Property preserving abstractions for the verification of concurrent systems. Form. Methods Syst. Des. **6**(1) (1995)
125. Lugiez, D., Niebert, P., Zennou, S.: A partial order semantics approach to the clock explosion problem of timed automata. Theor. Comput. Sci. **345**(1) (2005)
126. Lukoschus, B.: Compositional verification of industrial control systems: Methods and case studies. Ph.D. thesis, Universität zu Kiel, Technischen Fakultät der Christian–Albrechts (2005)
127. Luttik, B.: What is algebraic in process theory? Electronic Notes in Theoretical Computer Science **162**, 227–231 (2006). Proceedings of the Workshop "Essays on Algebraic Process Calculi" (APC 25)
128. Mahony, B., Dong, J.: Blending Object-Z and Timed CSP: an introduction to TCOZ. In: ICSE '98: Proceedings of the 20th international conference on Software engineering (1998)
129. Mandal, M., Asif, A.: Continuous and Discrete Time Signals and Systems. Cambridge University Press, Cambridge, USA (2007)
130. Manna, Z., Pnueli, A.: Temporal Verification of Reactive Systems: Safety (1995)
131. Mayer, R., Benjamin, P., Caraway, B., Painter, M.: A Framework and a Suite of Methods for Business Process Reengineering, Business Process Change: Reengineering, Concepts, Methods and Technologies, pp. 245–290. IdeaGroup Publishing, Hershey, USA (1995)
132. McMillan, K.: Symbolic Model Checking
133. McMillan, K.: A methodology for hardware verification using compositional model checking. Sci. Comput. Program. **37**(1-3) (2000)
134. McMillan, K.: The SMV System – version 2.5.4. Carnegie Mellon University, Pittsburg, USA (2000)
135. Meade, L., Rogers, K.: Selecting critical business processes: A case study. Engineering Management Journal **December** (2001)
136. Mendoza, L., Capel, M.: Consistency checking of UML composite structure diagrams based on trace semantics. In: Software Engineering in Progress – 2nd IFIP Central and East European Conference on Software Engineering Techniques (CEE-SET 2007) (2007)
137. Mendoza, L., Capel, M.: Algorithm proposal to automata generation from CCTL formulas. Technical report, University of Granada (2008)
138. Mendoza, L., Capel, M.: Procedure proposal to CSP+T process terms generation from automata. Technical report, University of Granada (2009)

139. Mendoza, L., Capel, M., Benghazi, K.: Checking behavioural consistency of UML–RT models through trace–based semantics. In: Proc. 9th International Conference on Enterprise Information Systems (ICEIS 2007), pp. 205–211. INSTICC Press, Setúbal, Portugal (2007)
140. Mendoza, L., Capel, M., Benghazi, K.: Towards the correctness verification of business processes modelled with UML. In: Proc. 13th Conference on Software Engineering and Databases (JISBD 2008), pp. 159–170. SISTEDES, Madrid, Spain (2008)
141. Mendoza, L., Capel, M., Pérez, M.: Compositional verification of business processes by model-checking. In: Proc. 8th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems (MSVVEIS 2010), pp. 60–69. INSTICC Press, Setúbal, Portugal (2010)
142. Mendoza, L., Capel, M., Pérez, M.: Compositional verification of business processes modelled with BPMN. In: Proc. 12th International Conference on Enterprise Information Systems (ICEIS 2010), vol. 3 – Information Systems Analysis and Specification, pp. 113–122. INSTICC Press, Setúbal, Portugal (2010)
143. Mendoza, L., Capel, M., Pérez, M., Benghazi, K.: Compositional model-checking verification of critical systems. In: Proc. 10th International Conference on Enterprise Information Systems (ICEIS 2008), pp. 213–225. INSTICC Press, Setúbal, Portugal (2008)
144. Mendoza, L., Marius, A., Pérez, M., Grimán, A.: Critical success factors for a customer relationship management strategy. Inf. Softw. Technol. **49**(8) (2007)
145. Metzler, B., Wehrheim, H., Wonisch, D.: Decomposition for compositional verification. In: ICFEM '08: Proceedings of the 10th International Conference on Formal Methods and Software Engineering (2008)
146. Miller, A., Donaldson, A., Calder, M.: Symmetry in temporal logic model checking. ACM Comput. Surv. **38**(3) (2006)
147. Milner, R.: A Calculus of Communicating Systems. LNCS 92. Springer–Verlag, Heidelberg (1980)
148. Milner, R.: Communicating and Mobile Systems: the Π–calculus (1999)
149. Moffat, N., Goldsmith, M.: Assumption—commitment support for csp model checking. J. Autom. Reason. **41**(3-4) (2008)
150. Moon, J., Lee, D., Park, C., Cho, H.: Transformation algorithms between bpel4ws and bpml for the executable business process. In: Enabling Technologies: Infrastructure for Collaborative Enterprises, 2004. WET ICE 2004. 13th IEEE International Workshops on, pp. 135–140 (2004)
151. Morimoto, S.: A Survey of Formal Verification for Business Process Modeling, LNCS 5102: Proc. 8th International Conference on Computational Science (ICCS 2008), pp. 514–522. Springer–Verlag, Berlin (2005)
152. nola, R.A.E.: Diccionario de la Lengua Española, 22nd edn.
153. OASIS: Web Services Business Process Execution Language Version 2.0. OASIS Open, Billerica, USA (2007)
154. Odeh, M., Kamm, R.: Bridging the gap between business models and system models. Information and Software Technology **45**(15), 1053–1060 (2003). Special Issue on Modelling Organisational Processes
155. Olderog, E.R.: Nets, Terms and Formulas: Three Views of Concurrent Processes and their Relationship. Cambridge University Press
156. OMG: MDA Guide Version 1.0.1. Object Management Group, Massachusetts, USA (2003)
157. OMG: UML Superstructure Specification – version 2.1.1. Object Management Group, Massachusetts, USA (2007)
158. OMG: Business Process Modeling Notation – version 1.2. Object Management Group, Massachusetts, USA (2009)
159. Opdahl, A., Henderson-Sellers, B., Barbier, F.: Ontological analysis of whole-part relationships in OO–models. Information and Software Technology **43** (2001)
160. Ortega, J., Márquez, J., Reina, A., Álvarez, J.: Definición y representación de reglas de transformación de un modelo independiente en otro específico de la plataforma. In: Actas IV Jornadas de Trabajo Dolmen, pp. 124–130. Grupo GEDES, Granada, España (2003)

161. Ouaknine, J., Worrell, J.: On Metric Temporal Logic and Faulty Turing Machines, LNCS 3921: Foundations of Software Science and Computation Structures, pp. 217–230. Springer Berlin, Heidelberg, Germany (2006)
162. Ould, M.: Business Process: Modelling and Analysis for Re-engineering and Improvement (1995)
163. Ouyang, C., Verbeek, E., van der Aalst, W., Breutel, S., Dumas, M., Hofstede, A.: Formal semantics and analysis of control flow in ws-bpel. Science of Computer Programming **67**(2–3), 162–198 (2007)
164. Papazoglou, M., Kratz, B.: Web services technology in support of business transactions. Service Oriented Computing and Applications **1**(1), 51–63 (2007)
165. Payne, A., Christopher, M., Clark, M., Peck, H.: Relationship Marketing for Competitive Advantage, second edn. Butterworth Heinemann, Oxford, UK (1999)
166. Peltz, C.: Web services orchestration and choreography. Computer **36**(10)
167. Pixley, C.: A computational theory and implementation of sequential hardware equivalence. In: DIMACS Workshop on Computer Aided Verification '90, pp. 293–320. DIMACS, Baltimore, USA (1990)
168. Place, P., Kang, K.: Safety-critical software: Status report and annotated bibliography. Technical report, Software Engineering Institute, Carnegie Mellon University (1993)
169. Plotkin, G.: An operational semantics for csp. In: A. Salwicki (ed.) Logics of Programs and Their Applications, Lecture Notes in Computer Science. Springer Berlin
170. Plotkin, G.: A structural approach to operational semantics. Journal of Logic and Algebraic Programming **60–61**, 17–139 (2004)
171. P.R., Niebert, P.: Partial order reduction on concurrent probabilistic programs. In: QEST '04: Proceedings of the The Quantitative Evaluation of Systems, First International Conference (2004)
172. Pressman, R.: Software Engineering: A Practitioner's Approach, 7th edn. McGraw-Hill series in computer science. McGraw-Hill Science/Engineering/Math, New York, USA (2009)
173. Queille, J.P., Sifakis, J.: Specification and verification of concurrent systems in cesar. In: Proceedings of the 5th Colloquium on International Symposium on Programming, pp. 337–351. Springer-Verlag, London, UK (1982)
174. Rabinovich, A.: On compositionality and its limitations. ACM Trans. Comput. Logic **8**(1) (2007)
175. Reed, G., Roscoe, A.: A timed model for communicating sequential processes. Theor. Comput. Sci. **58**(1–3) (1988)
176. Reichheld, F.: The Loyalty Effect. Harvard Business School Press, Boston, USA (1996)
177. de la Riva, C., Tuya, J.: Modular Model Checking of Software Specifications with Simultaneous Environment Generation, LNCS 3299: 2nd International Symposium on Automated Technology for Verification and Analysis (ATVA 2004), pp. 369–383. Springer–Verlag, Berlin (2004)
178. de la Riva, C., Tuya, J.: Automatic generation of assumptions for modular verification of software specifications. J. Syst. Softw. **79**(9) (2006)
179. Rivières, J.D., Wiegand, J.: Eclipse: A platform for integrating development tools. IBM Syst. J. **43** (2004)
180. Rockart, J.: Chief executives define their own data needs. Harvard Business Review **25**(2), 81–93 (1979)
181. Rodríguez, P., Alonso, J., Sánchez, J.: ¿cuál es la madurez que necesitarían los procesos para el desarrollo de sistemas de software crítico? Revista Española de Innovación, Calidad e Ingeniería del Software **1**(2) (2005)
182. de Roever, W., de Boer, F., Hannemann, U., Hooman, J., Lakhnech, Y., Poel, M., Zwiers, J.: Concurrency Verification: Introduction to Compositional and Noncompositional Methods, *Cambridge Tracts in Theoretical Computer Science*, vol. 54. Cambridge University Press, Cambridge, UK (2001)
183. de Roever, W.P., Langmaack, H., (Eds.), A.P.: Compositionality: The Significant Difference. LNCS 1536. Springer–Verlag, Heidelberg (1998)

184. Roscoe, A.: The Theory and Practice of Concurrency. Prentice–Hall International Ltd., Hertfordshire UK (1997)
185. Roscoe, A., Wu, Z.: Verifying Statemate Statecharts Using CSP and FDR, LNCS 4260: Formal Methods and Software Engineering, pp. 324–341. Springer Berlin, Heidelberg, Germany (2006)
186. Ruf, J.: RAVEN: Real–time analyzing abd verification environment. Journal of Universal Computer Science **7**(1), 89–104 (2001)
187. Rüf, J., Kropf, T.: Symbolic model checking for a discrete clocked temporal logic with intervals. In: Proceedings of the IFIP WG 10.5 International Conference on Correct Hardware Design and Verification Methods (1997)
188. Rüf, J., Kropf, T.: Modeling and checking networks of communicating real–time processes, LNCS 1703: Correct Hardware Design and Verification Methods (CHARME), pp. 256–279. Springer–Verlag, Berlin (1999)
189. Schmidt, D., Steffen, B.: Program analysis as model checking of abstract interpretations. In: SAS '98: Proceedings of the 5th International Symposium on Static Analysis (1998)
190. Schneider, S.: Concurrent and Real–Time Systems – The CSP Approach. John Wiley & Sons, Ltd., Chichester, England (2000)
191. Schriber, T.: Fundamentals of Flowcharting (1980)
192. Schulmeyer, G., MacKenzie, G.: Verification and Validation of Modern Software-Intensive Systems. Prentice-Hall, Inc. (2000)
193. Selic, B., Rumbaugh, J.: UML for Modeling Complex Real–Time Systems. ObjecTime Technical Report. ObjecTime, New York (1998)
194. Sipma, H., Uribe, T., Manna, Z.: Deductive model checking. Formal Methods in System Design **15**(1), 49–74 (1999)
195. Sistla, A., Gyuris, V., Emerson, E.: Smc: a symmetry-based model checker for verification of safety and liveness properties. ACM Trans. Softw. Eng. Methodol. **9**(2) (2000)
196. Sommerville, I.: Software Engineering 8, 8th edn. International Computer Science Series. Addison-Wesley, Massachusetts, USA (2006)
197. Spitznagel, B., Garlan, D.: A compositional formalization of connector wrappers. In: ICSE'03: Proceedings of the 25th International Conference on Software Engineering (2003)
198. Sue, P., Morin, P.: A strategic framework for CRM. Tech. rep.
199. Taylor, P.: Practical Foundations of Mathematics. Cambridge Studies in Advanced Mathematics, vol. 59. Cambridge University Press, Cambridge, USA (1999)
200. TC 176/SC 1: Quality management systems – Fundamentals and vocabulary. ISO/IEC Standard 9000. International Organization for Standardization, Geneva, Switzerland (2005)
201. Thakkar, S., Ambite, J., Knoblock, C.: Composing, optimizing, and executing plans for bioinformatics web services. The VLDB Journal **14**(3), 330–353 (2003)
202. Thompson, B.: What is CRM? Tech. rep.
203. Touati, H., Savoj, H., Lin, B., Brayton, R., Sangiovanni-Vincentelli, A.: Implicit state enumeration of finite state machines using bdd's. In: ICCAD-90: 1990 IEEE International Conference on Computer-Aided Design. Digest of Technical Papers., pp. 130–133 (1990)
204. Valmari, A.: Stubborn sets for reduced state space generation. In: Proceedings of the 10th International Conference on Applications and Theory of Petri Nets (1991)
205. Verbeek, H.: Verification of wf-nets. Ph.D. thesis, Technische Universiteit Eindhoven
206. Verbeek, H., van der Aalst, W., Kumar, A.: Xrl/woflan: Verification and extensibility of an xml/petri-net-based language for inter-organizational workflows. Information Technology and Management **5**(1), 65–110 (2004)
207. Verbeek, H., Basten, T., van der Aalst, W.: Diagnosing workflow processes using woflan. The Computer Journal **44**(4), 246–279 (2001)
208. Žic, J.: CSP+T: a formalism for describing real–time systems. Ph.D. thesis, University of Sydney, Basser Department of Computer Science (1991)
209. Žic, J.: Time–constrained buffer specifications in CSP+T and Timed CSP. ACM Transaction on Programming Languages and Systems **16**(6), 1661–1674 (1994)

210. Wang, F., Schmidt, K.: Symmetric symbolic safety-analysis of concurrent software with pointer data structures. In: FORTE '02: Proceedings of the 22nd IFIP WG 6.1 International Conference Houston on Formal Techniques for Networked and Distributed Systems (2002)
211. Wehrheim, H., Wonisch, D.: Compositional csp traces refinement checking. Electron. Notes Theor. Comput. Sci. **250**(2) (2009)
212. Wodtke, D., Weissenfels, J., Weikum, G., Kotz, A., Muth, P.: The mentor workbench for enterprise-wide workflow management. SIGMOD Rec. **26**(2) (1997)
213. Wong, P.: Towards a unified model for workflow processes. In: 1st Service-Oriented Software Research Network (SOSoRNet) Workshop. Manchester, UK (2006)
214. Wong, P., Gibbons, J.: Formalisations and applications of bpmn. Science of Computer Programming
215. Wong, P., Gibbons, J.: A Process-Algebraic Approach to Workflow Specification and Refinement, LNCS 4829: Software Composition, pp. 51–65. Springer Berlin, Heidelberg, Germany (2007)
216. Wong, P., Gibbons, J.: A process semantics for bpmn. LNCS (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **5256 LNCS**, 355–374 (2008)
217. Wong, P., Gibbons, J.: A relative timed semantics for BPMN. In: Proceedings of 7th International Workshop on the Foundations of Coordination Languages and Software Architectures, ENTCS (2008)
218. Wong, P., Gibbons, J.: A relative timed semantics for BPMN. In: Proceedings of 7th International Workshop on the Foundations of Coordination Languages and Software Architectures, ENTCS (2008)
219. Workflow Management Coallition (WfMC): Workflow Management Coallition Standard - Terminology and Glossary Issue 3.0. WfMC
220. Wynn, M., Verbeek, H., van der Aalst, W., ter Hofstede, A., Edmond, D.: Business process verification - finally a reality! Business Process Management Journal **15**(1), 74–92 (2009)
221. Yan-Ling, X., Fu-Yuan, X., Wen-Bo, H.: Business process reengineering based on idef methods. In: Information Reuse and Integration, 2004. IRI 2004. Proceedings of the 2004 IEEE International Conference on, pp. 265–270 (2004)
222. Yeh, W.J., Young, M.: Compositional reachability analysis using process algebra. In: TAV4: Proc. of the Symposium on Testing, Analysis, and Verification (1991)
223. Yeung, W., Leung, K., Wang, J., Dong, W.: Modelling and model checking suspendible business processes via statechart diagrams and CSP. Science of Computer Programming **65**(1), 14–29 (2007). Special Issue on: Increasing Adequacy and Reliability of EIS
224. Yeung, W., Schneider, S.: Design and verification of distributed recovery blocks with csp. Form. Methods Syst. Des. **22**(3) (2003)
225. Yourdon, E.: Modern Structured Analysis, 2nd ed. edn. (2000)
226. Zwiers, J.: Compositionality, Concurrency, and Partial Correctness: Proof Theories for Networks of Processes, and Their Relationship. LNCS 321. Springer–Verlag, Heidelberg (1989)