DTO. DE COMPUTACION
INFOTECA
F.C.E. y N. - U.B.A

Tesis

Licenciatura en Ciencias de la Computación

Monitoreo de Sistemas Temporizados especificados con Grafos de Diagramas de Secuencia (MSC-Graphs)

Sebastián BLAUSTEIN
Libreta universitaria: 069/96 – **e-mail:** sblauste@dc.uba.ar

Fernando OLIVETO
Libreta universitaria: 082/97 – **e-mail:** foliveto@dc.uba.ar

Director: Dr. Víctor BRABERMAN
e-mail: vbraber@dc.uba.ar

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Diciembre 2002

Indice

1 Abstract ... 4
2 Introducción ... 5
3 Sintaxis y Semántica ... 9
3.1 Conceptos teóricos ... 9
3.2 Sintaxis Abstracta ... 10
3.2.1 Sintaxis de un bMSC sin tiempos ... 10
3.2.2 Agregado de restricciones de tiempo a un bMSC sin tiempos ... 11
3.2.3 Sintaxis de un MSC-Graph ... 14
3.3 Semántica ... 15
3.3.1 Semántica de bMSC sin tiempos ... 15
3.3.2 Semántica de bMSC ... 18
3.3.3 Semántica de MSC-Graph ... 21
3.4 Sintaxis Concreta ... 24
3.4.1 Introducción a la sintaxis concreta de bMSC ... 24
3.4.2 Introducción a la sintaxis concreta de MSC-Graph ... 25
3.4.3 Introducción a la sintaxis concreta de log de ejecución ... 25
4 Solución algorítmica a los problemas de verificación ... 26
4.1 Notación ... 26
4.2 Pasos a seguir ... 26
4.3 Monitoreo de bMSCs sin tiempos ... 26
4.3.1 Definiciones ... 27
4.3.2 Analizando un bMSC sin tiempos ... 27
4.3.3 Algoritmo ... 28
4.3.4 Prueba de corrección ... 28
4.4 Monitoreo de MSC-Graphs sin tiempos ... 30
4.4.1 Definiciones ... 30
4.4.2 Analizando MSC-Graphs sin tiempos ... 31
4.4.3 Algoritmo ... 37
4.4.4 Prueba de corrección ... 39
4.5 Monitoreo de MSC-Graphs ... 41
4.5.1 Definiciones ... 41
4.5.2 Analizando MSC-Graphs ... 41
4.5.3 Algoritmo ... 43
4.5.4 Prueba de corrección ... 43
4.5.5 Algoritmos auxiliares ... 44
5 Implementación ... 53
5.1 Arquitectura ... 53
5.1.1 Arquitectura conceptual ... 53
5.1.2 Arquitectura de implementación ... 54
5.2 Diseño y estructuras de representación ... 57
5.2.1 Diseño de la herramienta ... 57
5.2.2 Estructuras de representación ... 59
5.3 Métricas de la herramienta MonMSC ... 60
5.4 Manual de uso de la herramienta MonMSC ... 61
5.4.1 Instalación de MonMSC ... 61
5.4.2 Modo de uso de MonMSC ... 61
5.4.3 Modo de uso de Telelogic TAU ... 63
5.5 Metodología de desarrollo ... 68
5.5.1 Diseño e implementación ... 68
5.5.2 Configuration Management y administración de Artifacts ... 68
5.5.3 Testing ... 69
5.5.4 Javadoc ... 70
6 Casos de Estudio ... 71
6.1 Cajero automático (ATM) ... 71
6.1.1 Informe del resultado obtenido ... 73
6.2 Expendedora de productos (VM) ... 78
6.2.1 Informe del resultado obtenido ... 80
7 Conclusiones y aportes ... 87
8 Trabajos a futuro ... 88
9 Agradecimientos ... 90

10 Referencias 91
11 Anexo A 93
11.1 Operaciones Auxiliares 93
11.1.1 Prefijos y Prefijar 93
11.2 Definición de la Sintaxis Concreta 93
11.2.1 Introducción al metalenguaje de la gramática textual 93
11.2.2 Definición de la sintaxis concreta de bMSC 94
11.2.3 Definición de la sintaxis concreta de MSC-Graph 97
11.2.4 Definición de la sintaxis concreta de log de ejecución 99

1 Abstract

Message Sequence Charts (MSCs) es una notación gráfica y formal (sintaxis y semántica definibles) usualmente utilizada para describir posibles escenarios en la comunicación y coordinación de componentes distribuidas. Es frecuentemente utilizada por los ingenieros de sistemas de telecomunicaciones (estandarizada por la ITU como Z.120) y, con ciertas modificaciones forma parte del estándar de facto UML.

Dado que en los MSCs el comportamiento de comunicación se presenta de una manera muy intuitiva y transparente, especialmente en la representación gráfica, el lenguaje se aprende, se utiliza y se interpreta con facilidad, en relación con otros lenguajes. Puede utilizarse para el soporte de metodologías de especificación, diseño, simulación, testing y documentación de sistemas.

Los MSCs no fueron diseñados para un dominio de aplicación único, sino que poseen un amplio campo de aplicación. Uno de ellos es la especificación general del comportamiento de comunicación de los sistemas en tiempo real, en particular, los sistemas de conmutación de telecomunicaciones. Mediante MSCs pueden especificarse sistemas seleccionados, primordialmente los casos normales. Los casos no normales, que comprenden comportamientos excepcionales, se pueden describir a partir de los normales. De este modo, pueden utilizarse MSCs para la especificación de requerimientos, especificación de interfaces, simulación y validación, especificación de pruebas y documentación de sistemas en tiempo real.

También pueden utilizarse junto con otros lenguajes de especificación, en particular el lenguaje de especificación y descripción SDL (*Specification and Description Language*) . MSCs y SDL se complementan mutuamente. SDL provee una descripción completa del comportamiento de procesos individuales, pero no hay una descripción directa de la comunicación entre estas entidades. Por otro lado, MSCs sí provee esta comunicación.

MSC-Graphs permite especificar infinitos escenarios y diferentes alternativas de selección. Consiste en un grafo, donde cada nodo es un MSC que contiene un conjunto de entidades que intercambian mensajes a través de eventos de envío y recepción, más restricciones de tiempo.

Los sistemas comerciales actuales que se utilizan para diseñar MSCs no poseen como funcionalidad el chequeo de comportamiento observado del software contra el especificado en MSCs. Por eso, esta funcionalidad constituye una herramienta práctica y poderosa que permitirá mejorar la calidad de los sistemas de tiempo real.

En nuestra tesis, desarrollamos a nivel teórico y práctico, una herramienta que a partir de logs de ejecuciones, analiza la conformidad de los mismos respecto al comportamiento especificado con MSC-Graphs.

La herramienta informa cubrimiento de escenarios, detecta y diagnostica anomalías en el orden de aparición y temporización de eventos con respecto a los escenarios especificados.
Constituye una potencial ayuda en los procesos de verificación de software distribuido de tiempo real.

La implementación de la herramienta se ha integrado con Telelogic TAU, una aplicación comercial CASE de uso extendido en el ámbito del software de telecomunicaciones, que genera MSC-Graphs bajo el estándae ITU Z.120.

2 Introducción

La tarea de encontrar errores de software es muy importante. A su vez, la gente que desarrolla métodos técnicos formales y herramientas, con frecuencia encuentra muy difícil la tarea de transferencia de tecnología a la industria de desarrollo de software. Una de las razones de esta aparente contradicción es que la investigación de métodos formales resulta frecuentemente en formalismos que son diferentes a aquellos utilizados por los diseñadores y desarrolladores de software. Sólo algunos de los formalismos, como lógica temporal o la notación Z [Z95], tuvieron éxito moderado fuera de la comunidad de desarrolladores de métodos formales. Sin embargo, la mayoría de las especificaciones de software se realiza informalmente, mientras el método más común utilizado para validar la confianza del software es el testing.

Una solución parcial a esta situación es el uso de formalismos visuales. Estos son supuestamente más intuitivos que los formalismos descriptos en forma textual, en el sentido de "una imagen es mejor que mil palabras". La descripción de software en diferentes maneras, desde distintas perspectivas y enfatizando diferentes aspectos, puede derivar en una mejor comprensión.

Otro grupo de formalismos son los estándares de ITU e ISO, incluyendo entre ellos *Specification and Description Language* (SDL) [SDL94] y LOTOS [BB87]. Ambos formalismos tienen una representación visual, como así también textual. El presente trabajo se focalizará en Message Sequence Chart (MSC) de ITU. Este formalismo es el más frecuentemente usado en el diseño de protocolos de comunicación y fue desarrollado en conjunto con el estándar SDL. MSCs permite la abstracción de partes del sistema que son menos relevantes para una vista en particular, para que sean perfectamente utilizados en la especificación de funcionalidades requeridas. MSC ha sido conocido por mucho tiempo como Sequence Diagram, Message Sequence Diagram o Timing Diagram. En los últimos años, se observa un gran crecimiento en el interés del desarrollo de herramientas y algoritmos que manipulan diseños basados en MSC ([AY99], [AEY00], [AHP96]), motivados por la integración de estos formalismos en *Unified Modeling Language* (UML) [BJR97].

El estándar visual y textual de la notación Z.120 de ITU, permite representar un escenario de ejecución simple, como así también una colección de escenarios, incluyendo alternativas y repeticiones. Esto se logra con la notación MSC-Graph, que consiste en un grafo, donde cada nodo contiene un MSC básico. El comportamiento de un sistema puede seguir los caminos de un grafo, comenzando desde el nodo inicial del mismo.

MSCs nació en el campo de las telecomunicaciones. Es un medio utilizado para la visualización de ejecuciones seleccionadas de sistemas concurrentes de telecomunicación. Puede ser entendido como un lenguaje de especificación especializado en el intercambio de mensajes entre entidades (procesos, servicios) y su ambiente, pero su principal ventaja es su clara disposición gráfica en su representación visual, que permite una comprensión intuitiva del comportamiento del sistema descripto.

MSCs es un lenguaje internacional estándar de ITU y consta de tres recomendaciones (1992, 1996 y 1999). La recomendación Z.120 [MSC92] introduce elementos básicos del lenguaje como instancias, mensajes, ambientes, timers, creación y terminación de procesos. En cambio, la recomendación Z.120 [MSC96] se concentra en construcciones estructurales del lenguaje, esencialmente composición y orientación a objetos en documentos MSC. Por último, la recomendación Z.120 [MSC99] es una continuación natural de la versión de 1996 a la que se agregaron los conceptos de datos y flujo de control, entre otros.

En forma simple, un MSC básico (bMSC) representa el intercambio deseado de mensajes correspondiente a una ejecución de un sistema en un determinado escenario. A través de los escenarios se logra mostrar comportamientos del sistema en situaciones normales o excepcionales.

Las principales características de MSCs son las siguientes:

- Un bMSC es un escenario que describe el orden en que tienen lugar las comunicaciones y otros eventos. Además, permite imponer restricciones a los valores de datos transmitidos y los tiempos de ocurrencia de los eventos.
- Soporta especificaciones completas e incompletas. Tiene la posibilidad de describir comportamientos incompletos utilizados en análisis efectuados en fases iniciales y para fines de documentación.
- Es un lenguaje gráfico. Los diagramas bidimensionales dan una visión de conjunto del comportamiento de procesos en comunicación. La forma textual del bMSC está prevista principalmente para el intercambio entre herramientas y como una base para el análisis formal automático.
- Es un lenguaje formal y práctico. Se utiliza en todo el proceso de ingeniería. Su campo de aplicación se extiende desde el análisis de dominio y la generación de ideas, pasando por las fases de formulación de

requerimientos y diseño, hasta las pruebas. MSCs se utiliza de manera diferente en las diversas fases, y es importante que este lenguaje tenga una semántica formal, así como una apariencia intuitiva.

- Tiene un amplio campo de aplicación. No está consagrado a un solo dominio de aplicación. Nació en el ámbito de las telecomunicaciones y actualmente se lo utiliza en la notación UML.
- Soporta el diseño estructurado. Escenarios simples (descritos por gráficos de bMSCs) pueden combinarse para formar especificaciones más completas por medio de MSC-Graphs. Los MSCs se reúnen en un documento MSC. Un diseño modular de escenarios está soportado por mecanismos para descomposición y reutilización.
- Se utiliza a menudo con otros métodos y lenguajes. Su definición permite la validación formal y automatizada de un bMSC con respecto a un modelo descrito en un lenguaje diferente. MSCs puede utilizarse, por ejemplo, con SDL y TTCN (*Tree and Tabular Combined Notation*) [MSC99].
- Un escenario especificado a través de un bMSC se interpreta usualmente de forma tal que la implementación debe, al menos, presentar el comportamiento expresado en el escenario. Otras interpretaciones son también posibles. Un bMSC puede utilizarse, por ejemplo, para especificar escenarios desautorizados.

Los MSC-Graphs han agregado otras características utilizadas para ampliar el poder expresivo del lenguaje en forma conveniente. En particular, permiten el reuso de los bMSCs y combinarse usando operaciones como la concatenación o la selección. Todas estas características fueron incorporadas en el estándar internacional Z.120.

Hoy en día, los MSCs y otros formalismos son utilizados para especificar requerimientos. Dichas especificaciones son compatibles naturalmente con métodos de diseño orientado a objetos y soportadas por casi todas las metodologías y notaciones modernas de desarrollo de software, como por ejemplo SDL, Room [SGW94] y UML.

Actualmente existen varias herramientas comerciales (como Rational Rose Real-Time, Rational Rose Enterprise Ed., Rhapsody, Telelogic Tau, RTDev Studio) que se utilizan para diseñar MSCs, pero no presentan la característica de chequear el comportamiento observado del software contra el especificado en MSCs.

De esta manera, crearemos una nueva funcionalidad: el monitoreo de sistemas utilizando sus especificaciones con MSC-Graphs, con el objetivo de detectar comportamientos inesperados (no especificados en el MSC-Graph). Esto permitirá, por ejemplo, mejorar la calidad de los sistemas de tiempo real.

Por ejemplo, en la Figura 1 se puede visualizar la especificación MSC-Graph de un cajero automático (ATM), conocida en la literatura y obtenida de [BL97]. La misma representa el comportamiento correcto y completo de un ATM. Cada nodo del MSC-Graph tiene asociado un bMSC y varios nodos pueden tener asociado el mismo bMSC.

En la Figura 2 se pueden observar algunos bMSCs asociados a los nodos. En el bMSC *GetBalance* correspondiente a uno de los nodos, se describe como un usuario solicita un balance de saldo al ATM, el ATM solicita el balance al banco, el banco le responde, el ATM imprime el resultado y luego despliega las opciones al usuario.

Todo intercambio de mensajes entre el usuario y el ATM tiene una restricción de [0, 2000] unidades de tiempo (en el diagrama se representa como milisegundos, una milésima de unidad de tiempo), que representa las unidades de tiempo como mínimo y máximo que deberán suceder entre el envío y la recepción de cada mensaje intercambiado.

A su vez en el bMSC *GetBalance*, entre la solicitud del balance por parte del ATM y la recepción de lo solicitado desde el banco, deberán suceder entre [0, 10000] unidades de tiempo. Lo mismo sucede en la recepción del balance por parte del ATM desde el banco con [3000, 5000] unidades de tiempo, y con el envío de la impresión y luego de las opciones por parte del ATM con [3000, 5000] unidades.

Además en en bMSC *ProcessPin*, antes de que el ATM verifique el pin ingresado contra el banco, se setea un Timer con una duración de 10000 unidades de tiempo. El mismo se resetea al recibir la respuesta de validación del banco, en el bMSC *GetOption*.

Se supone la siguiente secuencia de etiquetas con su tiempo respectivo, correspondiente a una ejecución del ATM:

```
S(CARD) - 0, R(CARD) - 1, S(REQ_PIN) - 2, R(REQ_PIN) - 3, S(ENTER_PIN) - 6, R(ENTER_PIN) - 7,
S(VERIFY) - 7, R(VERIFY) - 8, S(PROCESSING) - 8, R(PROCESSING) - 9, S(VALID) - 8, R(VALID) - 10,
S(OPTION) - 11, R(OPTION) - 12, S(STATEMENT) - 12, R(STATEMENT) - 12, S(REQ_BALANCE) - 13,
R(REQ_BALANCE) - 14, S(BALANCE) - 16, R(BALANCE) - 18, S(PRINT_STMT) - 22, R(PRINT_STMT) - 23,
S(OPTION) - 27, R(OPTION) - 28.
```

donde S(x) representa el evento Send del mensaje x y R(x) el evento de Receive del mismo.
El problema que se quiere resolver, es verificar si la secuencia de etiquetas temporizada corresponde a un comportamiento válido del sistema ATM especificado con MSCs.

En el ejemplo del ATM, la misma corresponde al camino formado por la secuencia de nodos *StartTrans*, *GetPin*, *ProcessPin*, *GetOption* y *GetBalance* del MSC-Graph. Sin embargo, también se requieren validar todas las restricciones de tiempo presentes en el camino, para cumplir con los requisitos que se propondrán a lo largo del presente trabajo.

Todos los temas se explicarán con profundidad en el trabajo y permitirán al lector comprender la solución al problema.

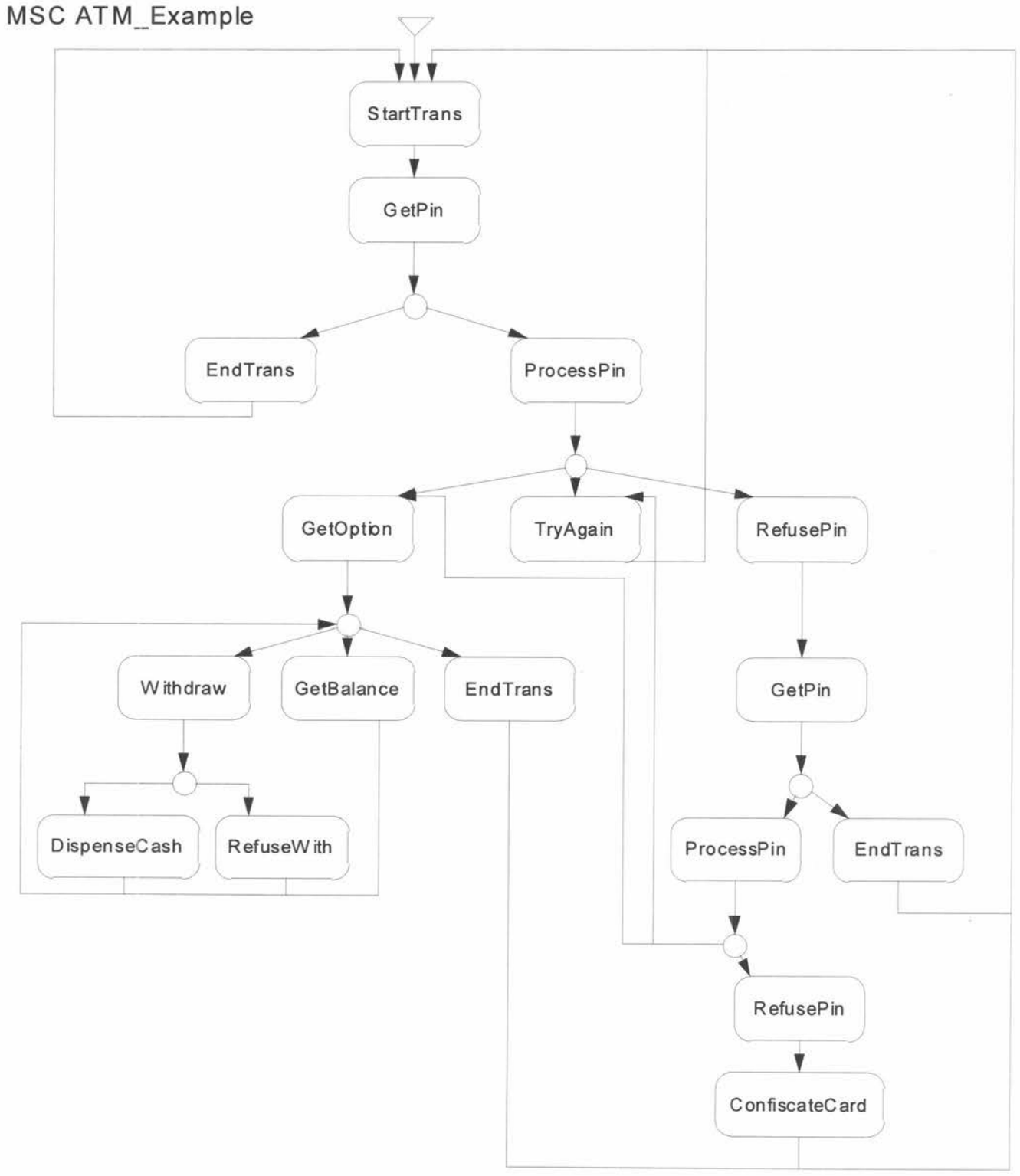


Figura 1

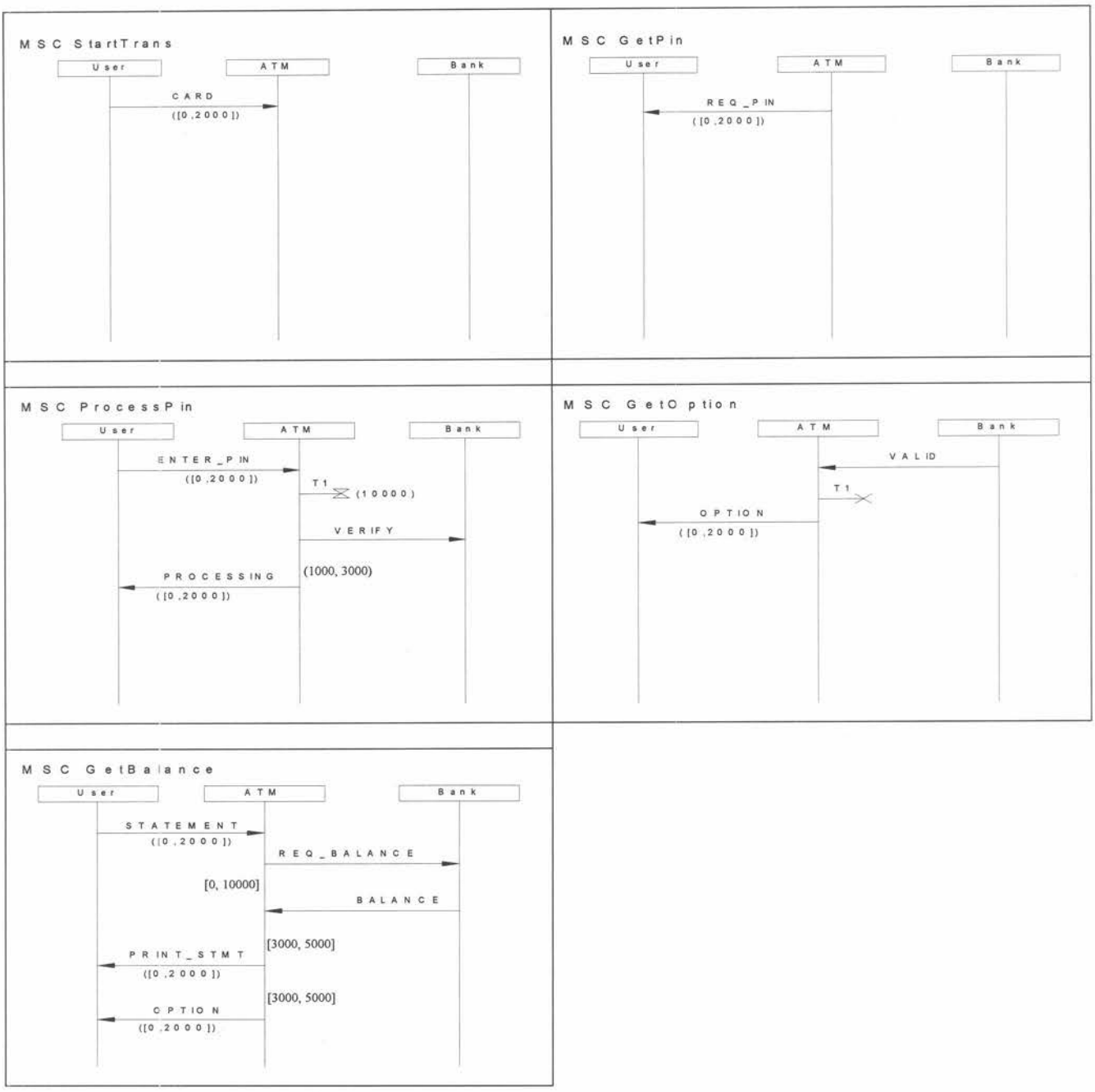


Figura 2

El presente trabajo, podrá ser utilizado como un oráculo cuando se lo utilice para testear requerimientos no funcionales, como así también para monitorear protocolos en tiempo real. El objetivo final consiste en desarrollar a nivel teórico y práctico un algoritmo que maneje estos tipos de detecciones para MSCs producidos por herramientas comerciales conocidas.

Un primer paso consistirá en definir formalmente la sintaxis y semántica de los bMSCs y MSC-Graphs, con el agregado formal de restricciones de tiempo. De esta manera la semántica de los bMSCs y MSC-Graphs se definirá en términos de un lenguaje formal temporizado (secuencias aceptables de eventos con Timestamps).

Luego se definirán los algoritmos que recibirán como entrada la especificación de sistemas en MSC-Graphs y un log de comportamiento de ejecuciones. Su objetivo será detectar si el log responde a un comportamiento aceptable. Estos algoritmos serán probados correctos y también se describirán los algoritmos para las operaciones más representativas. Se detallará la arquitectura de la herramienta desarrollada, así como un diseño de la misma. Se desarrollarán casos de estudio mostrando datos relevantes sobre los mismos y los resultados obtenidos al ser ejecutados en la herramienta.

Además se agregará la funcionalidad de generar reportes de cubrimiento sobre la especificación MSC-Graph, de acuerdo al log de ejecución recibido.

3 Sintaxis y Semántica

3.1 Conceptos teóricos

Antes de comenzar con las definiciones y soluciones propuestas a los problemas planteados, se recuerda una serie de conceptos que serán importantes en el presente trabajo y que permitirán al lector interpretar en forma correcta la sintaxis y semántica propuestas.

Orden Parcial (P.O.)

Un orden parcial se define por: (E, <), donde E es un conjunto de elementos, < una relación de orden y se cumple:

1. $< \subseteq E \times E$.
2. $\neg\, a < a$ (irreflexiva)
3. $a, b, c \in E: a < b \wedge b < c \Rightarrow a < c$ (transitiva)
4. $a, b \in E: a < b \Rightarrow \neg\, b < a$ (asimétrica)

Configuración

Dado un P.O. $P = (E, <)$, $C = (E', <')$ es una configuración de P si y sólo sí:

1. C es un orden parcial.
2. $E' \subseteq E \wedge <' \subseteq <$.
3. $\forall\, a, b \in E: a < b \wedge b \in E' \Rightarrow a \in E'$.

Dado un P.O., una **configuración** es un subconjunto cerrado[1] hacia atrás de elementos, tal que permite ver en forma gráfica las dependencias entre los mismos. Es decir, que una **configuración** es un prefijo del P.O.

Conjunto de Configuraciones de un Orden Parcial

Se define un conjunto de configuraciones de un P.O. P como conf(P) = {C / C es una configuración de P}.

[1] Cerrado: Sea C una configuración, si $a \in C \wedge b < a \Rightarrow b \in C$

3.2 Sintaxis Abstracta

En esta sección se especifica la sintaxis considerada válida para Message Sequence Charts básicos, Message Sequence Charts básicos con restricciones de tiempo y MSC-Graphs. La definición captura la esencia de bMSCs y MSC-Graphs del estándar Z.120 ITU [MSC96], y es análoga a las definiciones encontradas en [AHP96] y [AY99].

3.2.1 Sintaxis de un bMSC sin tiempos

Se define un bMSC sin tiempos como un MSC básico sin restricciones de tiempo. Para formalizar un bMSC sin tiempos y permitir su análisis, un ejemplo puede verse en la Figura 3.

Este contiene tres instancias asincrónicas, nombradas de izquierda a derecha i_1, i_2, i_3. Por cada instancia i en el sistema, existe una línea vertical que define un orden visual local, denotado $<_i$, en todos los eventos que pertenecen a i. Cada evento puede ser de envío o recepción y corresponde a una instancia específica. Los eventos de envío o recepción de mensajes son etiquetados por s_1, s_2, s_3, s_4, r_1, r_2, r_3 y r_4. Por cada evento de envío, existe un evento de recepción que le corresponde, y viceversa. El orden real de ocurrencia de dos eventos cualesquiera en un bMSC sin tiempos, puede o no corresponder al orden que muestra la figura. Los mensajes intercambiados entre las instancias son representados por flechas. La cola de cada flecha corresponde al evento de envío del mensaje y la cabeza a su recepción. Cada flecha es identificada con un nombre, que es la fuente para las etiquetas de Send y de Receive del mensaje en cuestión.

Se procede a definir formalmente un bMSC sin tiempos, en forma similar a [AHP96] y [AY99] con agregados de etiquetas.

Un bMSC sin tiempos $M = (I, E, g, f, \Sigma, l, \{<_i\})$ es una tupla conformada por:

- ***Instancias***: I es un conjunto finito de instancias.
- ***Eventos***: E es un conjunto finito de eventos formado por dos subconjuntos. Uno de eventos de envío S y uno de eventos de recepción R.
- ***Instancia de evento***: g es una función que relaciona cada evento de E a una instancia en I. El conjunto de eventos pertenecientes a una instancia i es llamado E_i.
- ***Ejes***: La función biyectiva $f: S \rightarrow R$ asocia cada evento de envío s con un único evento de recepción $f(s)$, y cada evento de recepción r con un único evento de envío $f^{-1}(r)$.
- ***Alfabeto***: Σ es un alfabeto de etiquetas de la forma S(x) o R(x), donde x es el nombre asociado a un eje.
- ***Etiquetado***: l es una función total que etiqueta cada evento de E con un símbolo del alfabeto Σ. Nota: Puede suceder que diferentes eventos estén asociados a una misma etiqueta.
- ***Orden Visual***: Para cada instancia i existe un orden local total $<_i$ sobre los eventos E_i que corresponde al orden en que los eventos aparecen en el diagrama.
- ***Relación*** <: El Orden Visual junto con los ejes, definen la relación $< = [(\cup_i <_i) \cup \{ (s, f(s)) \mid s \in S\}]^*$. Esta relación debe cumplir la restricción de ser un orden parcial[2] sobre E. El orden parcial se denominará M.PO.

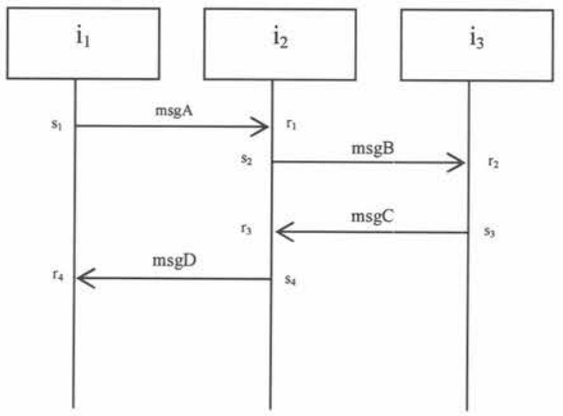


Figura 3

[2] Ver la sección 3.3.1 Semántica de bMSC.

La definición formal del bMSC sin tiempos de la Figura 3 se describe a continuación:

bMSC sin tiempos $M = (I, E, g, f, \Sigma, l, \{<_i\})$

$I = \{i_1, i_2, i_3\}$

$E = S \cup R = \{s_1, s_2, s_3, s_4\} \cup \{r_1, r_2, r_3, r_4\}$

$g = g(s_1) = i_1;\ g(r_1) = i_2;\ g(s_2) = i_2;\ g(r_2) = i_3;\ g(s_3) = i_3;\ g(r_3) = i_2;\ g(s_4) = i_2;\ g(r_4) = i_1$

$E_{i1} = \{s_1, r_4\};\ E_{i2} = \{r_1, s_2, r_3, s_4\};\ E_{i3} = \{r_2, s_3\}$

$f = f(s_1) = r_1;\ f(s_2) = r_2;\ f(s_3) = r_3;\ f(s_4) = r_4$

$f^{-1} = f^{-1}(r_1) = s_1;\ f^{-1}(r_2) = s_2;\ f^{-1}(r_3) = s_3;\ f^{-1}(r_4) = s_4$

$\Sigma = \{$ S(msgA), R(msgA), S(msgB), R(msgB), S(msgC), R(msgC), S(msgD), R(msgD) $\}$

$l = l(s_1) = S(msgA);\ l(r_1) = R(msgA);\ l(s_2) = S(msgB);\ l(r_2) = R(msgB);\ l(s_3) = S(msgC);\ l(r_3) = R(msgC);\ l(s_4) = S(msgD);\ l(r_4) = R(msgD)$

$<_{i_1} = \{s_1 < r_4\};\ <_{i_2} = \{r_1 < s_2 < r_3 < s_4\};\ <_{i_3} = \{r_2 < s_3\}$

$< = [<_{i_1} \cup <_{i_2} \cup <_{i_3} \cup \{(s_1, r_1); (s_2, r_2); (s_3, r_3); (s_4, r_4)\}]^*$

$= [\{s_1 < r_4\} \cup \{r_1 < s_2 < r_3 < s_4\} \cup \{r_2 < s_3\} \cup \{s_1 < r_1, s_2 < r_2, s_3 < r_3, s_4 < r_4\}]^*$

Nota: a lo largo del presente trabajo se utilizará el símbolo < para referirse a la relación "<" de un bMSC. Sin embargo, el símbolo < debe interpretarse de acuerdo a su contexto de uso: si se lo utiliza para relacionar dos números enteros se está refiriendo a la operación matemática menor, pero si se relacionan dos eventos se refiere a la relación "<" de un bMSC.

Restricción 1: Alcance de las etiquetas de los mensajes

$(\forall\ e_1, e_2 \in S \cup R \mid l(e_1) = l(e_2) \Rightarrow \mathbf{g(e_1) = g(e_2)})$

No se permite que existan eventos de mensaje con etiquetas similares, que tengan distinto origen y/o destino. Para salvar este problema, se debe realizar un renombramiento de los mensajes que tienen similar nombre y diferentes origen y/o destino.

En la Figura 4 se puede ver un ejemplo donde un problema sucede con la ocurrencia de la etiqueta S(msgA) (Send "msgA"). Se puede ver que tanto s_1 como s_2 se encuentran asociadas a la etiqueta S(msgA). Cuando se recibe en la secuencia de ejecución S(msgA) como etiqueta, no es posible saber a qué instancia atribuir este mensaje, con lo que se genera un no-determinismo (se podría asociar tanto a s_1, como a s_2). Lo mismo sucederá con la etiqueta R(msgA). Para salvar este problema, se debe realizar un renombramiento de los mensajes que tienen similar nombre y diferente origen y/o destino. De esta manera, en lugar de tener dos veces msgA, existiría I_1_I_2.msgA y I_3_I_4.msgA.

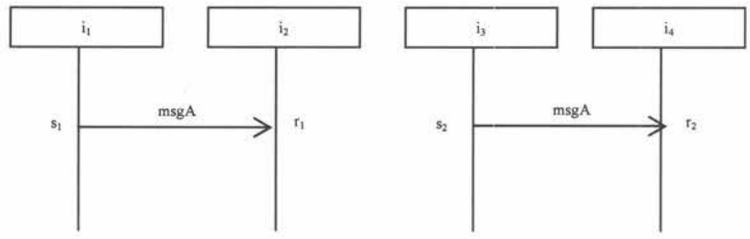


Figura 4

3.2.2 Agregado de restricciones de tiempo a un bMSC sin tiempos

Se define un bMSC como un MSC básico con restricciones de tiempo y se toma como base la definición de tiempos de [MSC96], [MSC98], [BL97] y [LL99].

Para formalizar un bMSC y permitir su análisis, un ejemplo puede verse en la Figura 5. El bMSC de la figura, se corresponde al bMSC sin tiempos de la Figura 3 analizado en la sección anterior, con el agregado de restricciones de tiempo y un nuevo mensaje. En él se pueden observar dos Timers y dos Delay Intervals.

El primer Timer *T1.1*, con *3* unidades de tiempo de duración, perteneciente a la instancia i_1, es seteado luego del evento s_1 y reseteado luego del evento r_4. Representa que el evento r_4 deberá ocurrir antes del reseteado del Timer. Esto quiere decir que después de seteado el Timer, el evento r_4 no podrá ocurrir más allá de *3* unidades de tiempo, en el peor de los casos. Por resetearse el Timer, el mismo no le aplica ninguna restricción de tiempo al evento r_5.

El segundo Timer *T3.1*, con *5* unidades de tiempo de duración, perteneciente a la instancia i_3 es seteado antes de recibir el evento r_2, y ocurre su timeout luego del evento s_3. Representa que ambos eventos r_2 y s_3 deberán ocurrir antes de

timeout del Timer. Esto quiere decir que después de seteado el Timer, los eventos r_2 y s_3 no podrán ocurrir más allá de *5* unidades de tiempo, en el peor de los casos. Por el timeout del Timer, el evento s_5 deberá suceder después de 5 unidades de tiempo desde el seteo del Timer.

El DI asociado a un eje o mensaje aparece sobre *msgA*, posee a *1* como mínimo (no incluido) y *2* como máximo. Esto significa que se espera que $1 < x \leq 2$ donde x es el intervalo de tiempo que podrá transcurrir entre s_1 y r_1.

El DI asociado a la velocidad de procesamiento aparece entre r_2 y $s_3 \in i_3$, posee a *2* como mínimo y *4* como máximo (no incluido). Esto significa que se espera que $2 \leq x < 4$ donde x es el intervalo de tiempo que podrá transcurrir entre r_2 y s_3.

Se trabajará con dos definiciones diferentes de sintaxis para expresar control de tiempo en bMSCs, a saber:

a. Timers

- Un Timer es asignado a una única instancia y no puede ser compartido entre diferentes instancias en un mismo bMSC.
- Se puede crear asignándole una duración *n* y un nombre T
- Se puede apagar reseteándolo a cero
- Puede ser observado por Timeout

b. Delay Intervals (DI)

- Los DI pueden expresar dos tipos de controles de tiempo:
  Asociados a un eje (mensaje): Se expresa como un mínimo y un máximo de tiempo sobre el mensaje representado por un eje. Representa el intervalo de tiempo permitido entre el evento de envío y recepción del mensaje asociado.
  Asociados a la velocidad de procesamiento: Expresa un intervalo de tiempo permitido entre dos eventos de mensaje consecutivos de una misma instancia.

- Un DI puede tener la forma [a, b]; [a, b); (a, b] o (a, b). Donde $a \in N$ es el mínimo; $b \in N \cup \{\infty\}$ es el máximo. Cuando $b = \infty$, el intervalo sólo puede ser de la forma [a, b) o (a, b).

- El mínimo y máximo del DI representan unidades de tiempo, a ser definidas por el contexto del problema a analizar. Por ejemplo, segundos, milisegundos, etc.

Es recomendable utilizar Delay Intervals en lugar de Timers cuando se quiere establecer restricciones de tiempo en un mismo bMSC, dado que permiten representar mejores controles entre eventos consecutivos en una misma instancia o de "envío – recepción" entre diferentes instancias.

En la Figura 5 se muestra un ejemplo de un bMSC, donde es posible visualizar algunas restricciones de tiempo:

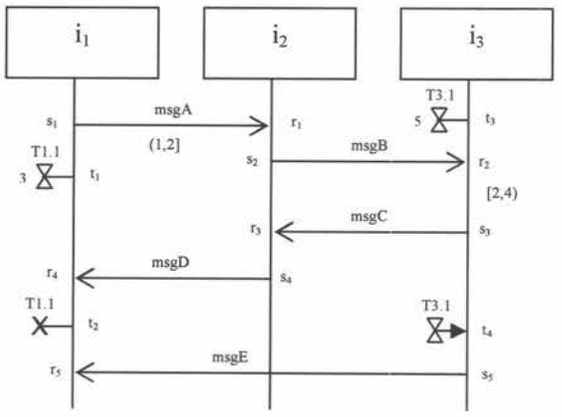


Figura 5

Se extiende la definición formal de un bMSC sin tiempos a bMSC, con la inclusión de eventos de Timer y Delay Intervals, a la siguiente notación.

Un bMSC $M_t = (I, E, g, f, \Sigma, l, \{<_i\}, T, D, tipoEv, OpTimer, GetTimer, Inst, Dur, IntDI, TipoDI, ParEvDI)$ es una tupla donde:

- $M = (I, E, g, f, \Sigma, l, \{<_i\})$ es el bMSC sin tiempos, definido con anterioridad, a excepción del conjunto de eventos E que se redefine a continuación.
- ***Eventos***: E es un conjunto finito de eventos formado por tres subconjuntos. Uno de eventos de envío S, uno de eventos de recepción R y uno de eventos de Timers θ. Posee tres operaciones:
  Tipo de Evento: tipoEv es una función que relaciona un evento a un subconjunto de evento determinado (S, R o θ).
  Operación de Timer: OpTimer es una función que relaciona un evento $e \in \theta$ a uno de las siguientes posibles operaciones: *OpTimer(e) = Set, Reset o Timeout.*
  Obtener un Timer: GetTimer es una función que relaciona un evento $e \in \theta$ a un Timer $t \in T$.
- ***Timers***: T es un conjunto finito de Timer. Posee dos operaciones:
  Instancia del Timer: Inst es una función que relaciona cada Timer $t \in T$ a una instancia $i \in I$. Como regla para asegurar que cualquier evento e de un Timer pertenece a la misma Instancia que el Timer, se pide: $Inst(GetTimer(e)) = g(e)$.
  Duración del Timer: Dur es una función que relaciona cada Timer $t \in T$ a una duración de tiempo $d \in N$, que resulta ser la última duración asignada al Timer con un evento de Set.
- ***Delay Intervals***: D es un conjunto finito de Delay Intervals. Posee tres operaciones:
  Intervalo: IntDI es una función que relaciona un Delay Interval a un intervalo de la forma *[a, b]; [a, b); (a, b] o (a,b).*
  Tipo Delay Interval: TipoDI es una función que relaciona un Delay Interval a un posible tipo: *"ej"* (asociado a eje o mensaje) o *"vp"* (asociado a velocidad de procesamiento).
  Par de Eventos: ParEvDI es una función que relaciona un Delay Interval *di* a un par de eventos $e_1, e_2 \in (S \cup R)$ tal que:
  - Si $TipoDI(di) = ej \Rightarrow (e_1, e_2)$ cumple: $f(e_1) = e_2$. El Delay Interval actúa sobre el eje determinado por los eventos e_1 y e_2.
  - Si $TipoDI(di) = vp \Rightarrow (e_1, e_2)$ cumple: $g(e_1) = g(e_2)$. Además $e_1 <_i e_2 \wedge (\neg\exists\, e_3 : e_3 \in (S \cup R) \mid e_1 <_i e_3 \wedge e_3 <_i e_2)$. El Delay Interval actúa sobre el intervalo determinado por los eventos de mensaje consecutivos e_1 y e_2.

Observación:
Un evento de Set sobre un Timer t, se dice *abierto* si no es seguido por un evento de Timeout o de Reset sobre el mismo t, en el orden visual del bMSC.
Un evento de Reset o Timeout sobre un Timer t, se dice *abierto* si no es precedido por un evento de Set sobre el mismo t, en el orden visual del bMSC.

La definición formal del bMSC de la Figura 5 se describe a continuación:

$M_t = (I, E, g, f, \Sigma, l, \{<_i\}, T, D, tipoEv, OpTimer, GetTimer, Inst, Dur, IntDI, TipoDI, ParEvDI)$

$\boldsymbol{I} = \{i_1, i_2, i_3\}$

$\boldsymbol{E} = S \cup R \cup \theta = \{s_1, s_2, s_3, s_4, s_5\} \cup \{r_1, r_2, r_3, r_4, r_5\} \cup \{t_1, t_2, t_3, t_4\}$

$\boldsymbol{tipoEv} = tipoEv(s_1) = tipoEv(s_2) = tipoEv(s_3) = tipoEv(s_4) = tipoEv(s_5) = S$
$tipoEv(r_1) = tipoEv(r_2) = tipoEv(r_3) = tipoEv(r_4) = tipoEv(r_5) = R$
$tipoEv(t_1) = tipoEv(t_2) = tipoEv(t_3) = tipoEv(t_4) = \theta$

$\boldsymbol{OpTimer} = OpTimer(t_1) = Set, OpTimer(t_2) = Reset, OpTimer(t_3) = Set, OpTimer(t_4) = Timeout$

$\boldsymbol{GetTimer} = GetTimer(t_1) = T1.1, GetTimer(t_2) = T1.1, GetTimer(t_3) = T3.1, GetTimer(t_4) = T3.1$

$\boldsymbol{g} = g(s_1) = i_1; g(r_1) = i_2; g(s_2) = i_2; g(r_2) = i_3; g(s_3) = i_3; g(r_3) = i_2; g(s_4) = i_2; g(r_4) = i_1; g(s_5) = i_3; g(r_5) = i_1;$
$g(t_1) = i_1; g(t_2) = i_1; g(t_3) = i_3; g(t_4) = i_3$

$\boldsymbol{E_{i1}} = \{s_1, t_1, r_4, t_2, r_5\}; \boldsymbol{E_{i2}} = \{r_1, s_2, r_3, s_4\}; \boldsymbol{E_{i3}} = \{t_3, r_2, s_3, t_4, s_5\}$

$\boldsymbol{f} = f(s_1) = r_1; f(s_2) = r_2; f(s_3) = r_3; f(s_4) = r_4; f(s_5) = r_5$

$\boldsymbol{f^{-1}} = f^{-1}(r_1) = s_1; f^{-1}(r_2) = s_2; f^{-1}(r_3) = s_3; f^{-1}(r_4) = s_4; f^{-1}(r_5) = s_5$

$\boldsymbol{\Sigma} = \{ S(msgA), R(msgA), S(msgB), R(msgB), S(msgC), R(msgC), S(msgD), R(msgD), S(msgE), R(msgE) \}$

$\boldsymbol{l} = l(s_1) = S(msgA); l(r_1) = R(msgA); l(s_2) = S(msgB); l(r_2) = R(msgB); l(s_3) = S(msgC); l(r_3) = R(msgC);$
$l(s_4) = S(msgD); l(r_4) = R(msgD); l(s_5) = S(msgE); l(r_5) = R(msgE)$

$\boldsymbol{<_{i1}} = \{s_1 < t_1 < r_4 < t_2 < r_5\}; \boldsymbol{<_{i2}} = \{r_1 < s_2 < r_3 < s_4\}; \boldsymbol{<_{i3}} = \{t_3 < r_2 < s_3 < t_4 < s_5\}$

$< = [<_{i1} \cup <_{i2} \cup <_{i3} \cup \{(s_1, r_1); (s_2, r_2); (s_3, r_3); (s_4, r_4); (s_5, r_5)\}]^*$
$= [\{s_1 < t_1 < r_4 < t_2 < r_5\} \cup \{r_1 < s_2 < r_3 < s_4\} \cup \{t_3 < r_2 < s_3 < t_4 < s_5\} \cup \{s_1 < r_1, s_2 < r_2, s_3 < r_3, s_4 < r_4, s_5 < r_5\}]^*$

$\boldsymbol{T} = \{T1.1, T3.1\}$

$\boldsymbol{Inst} = Inst(T1.1) = i_1, Inst(T3.1) = i_3$

$\boldsymbol{Dur} = Dur(T1.1) = 3, Dur(T3.1) = 5$

$\boldsymbol{D} = \{di_1, di_2\}$

IntDI = IntDI(di_1) = (1, 2], IntDI(di_2) = [2, 4)
TipoDI = TipoDI(di_1) = "ej", TipoDI(di_1) = "vp"
ParEvDI = ParEvDI(di_1) = (s_1, r_1), ParEvDI(di_1) = (r_2, s_3)

De aquí en adelante cuando se defina un bMSC con restricciones de tiempo, se hará de la siguiente forma $M_t = (I, E, g, f, \Sigma, l, \{<_i\}, T, D)$, no incluyendo las operaciones sobre los Eventos, Timers y Delay Intervals, de forma de facilitar la lectura.

3.2.2.1 Definiciones

La siguiente definición representa una condición importante que pueden tener las instancias de un bMSC.

Definición 1: Instancia Idle

Dado bMSC $M = (I, E, g, f, \Sigma, l, \{<_i\})$, una instancia i $\in I$ se dice *Instancia Idle* en M sii ($\neg\exists$ e: evento : e $\in E \wedge$ (tipoEv(e) = $S \vee$ tipoEv(e) = R) | g(e) = i).

3.2.3 Sintaxis de un MSC-Graph

Una manera natural de construir múltiples escenarios es utilizando MSC-Graphs. Un MSC-Graph es un grafo donde cada nodo tiene asociado un bMSC y los ejes corresponden a concatenaciones de los bMSCs.

Un ejemplo puede verse en la Figura 6. Este consiste en un nodo inicial M_i que apunta al primer nodo M_1, correspondiente al escenario MSC 1. Y el nodo M_1 es seguido por el nodo M_2, que corresponde al escenario MSC 2.

Se procede a definir formalmente un MSC-Graph, en forma similar a [AY99] y [BL97].

Un *MSC-Graph* $G = (B, V^I, N, suc, getbMSC)$ es una tupla con los siguientes componentes:

- ***bMSCs :*** B es un conjunto finito de bMSCs tal que para cualquier
  $M_t = (I, E, g, f, \Sigma, l, \{<_i\}, T, D) \in B$, $M_t' = (I', E', g', f', \Sigma', l', \{<_i\}', T', D') \in B$.
  Si $M_t \neq M_t' \Rightarrow$
  - $E \cap E' = \varnothing$
    No se repiten eventos en diferentes bMSCs de los nodos del MSC-Graph.
  - $I = I'$
    Las instancias son las mismas para cada bMSC de los nodos del MSC-Graph.
  - Si $(\exists e \in E, e' \in E', k \in (\Sigma \cap \Sigma') \mid l(e) = k \wedge l(e') = k) \Rightarrow$ ***g(e) = g(e')***
    Si existen dos eventos que macheen a la misma etiqueta en diferentes nodos, ambos deben ir a la misma instancia.
- ***Nodos:*** $V^I \cup N$ es un conjunto finito de nodos tales que:
  - V^I = Conjunto formado por un único nodo inicial
  - N = Conjunto de nodos intermedios
- ***Ejes:*** $suc \subset V \times V$ es una relación que refleja la conectividad del MSC-Graph G tal que todos los nodos en V son alcanzables desde el nodo inicial.
- ***bMSC del nodo:*** $getbMSC: N \rightarrow B$ es una función que relaciona cada nodo intermedio a un $bMSC \in B$.

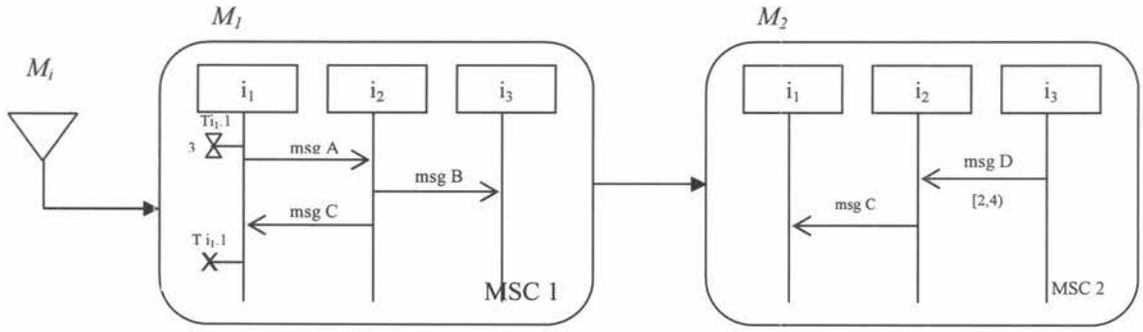


Figura 6

La definición formal del MSC-Graph de la Figura 6 se describe a continuación:

MSC-Graph G = (B, V^I, N, suc, getbMSC)
B = {MSC1, MSC2}
V^I = {M_i}
N = {M1, M2}
suc = {(M_i, M_1) ; (M_1, M_2)}
getbMSC = $M_1 \rightarrow$ MSC 1; $M_2 \rightarrow$ MSC 2

3.3 Semántica

La correcta interpretación semántica puede depender de muchos factores que no pueden ser estandarizados, como por ejemplo la particularidad de una arquitectura. Sin embargo, se especifica una serie de definiciones que permitirán unas razonables interpretaciones semánticas, análogas a las definiciones encontradas en [AY99], [LL99] y [GMP01].

Se definirán conceptos para la interpretación semántica de bMSCs sin tiempos, bMSCs y MSC-Graphs. Con ellos se formalizará la interpretación que se le deberá asociar a la sintaxis abstracta propuesta, de manera de evitar interpretaciones incorrectas. Es decir que a continuación se describirá la forma de interpretar y comprender los bMSCs sin tiempos, bMSCs y MSC-Graphs.

3.3.1 Semántica de bMSC sin tiempos

Recordando los conceptos iniciales introducidos, la interpretación semántica de un bMSC sin tiempos tiene que ver con el Orden Parcial del mismo. En particular, las configuraciones van a jugar un papel muy importante en la definición de la semántica y análisis de un bMSC, porque se definirá la semántica de las configuraciones parciales de un bMSC.

Para construir el grafo que muestra el orden parcial de un bMSC, es necesario utilizar el conjunto de relaciones locales totales $\{<_i\}$ (orden local total en cada instancia), en conjunto con los ejes (no es necesario realizar la clausura de esta relación).

A continuación se describe con un ejemplo el orden parcial del bMSC de la Figura 7 para afianzar los conceptos.

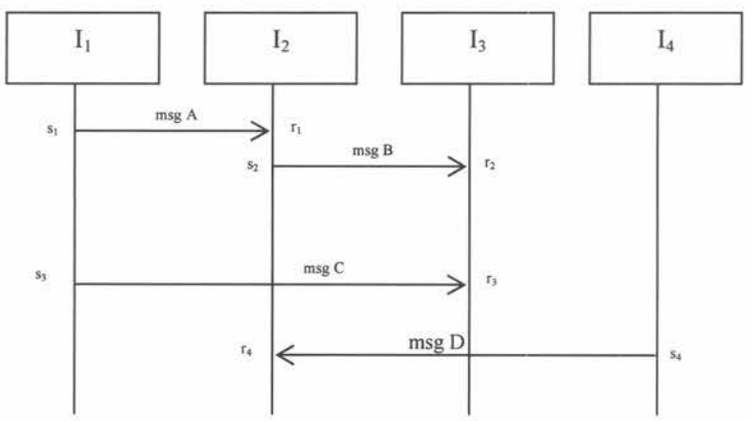


Figura 7

La Figura 8 despliega el grafo P.O. del bMSC sin tiempos correspondiente a la Figura 7. En él se puede observar claramente el orden parcial de los eventos:

- Los primeros eventos que pueden ocurrir en el bMSC son aquellos que no poseen precedencia alguna, en este caso el evento s_1 ó s_4.
- Luego, si sucede s_1, podrán ocurrir aquellos eventos que lo tienen solamente a él como precedencia. En este caso podrán ser r_1 ó s_3. En el caso que suceda s_4, no podrá ocurrir ningún evento que lo tenga como precedencia (r_4) porque requiere que otros eventos deban suceder primero.
- Suponiendo que sucedan s_1 y r_1, luego podrán suceder s_3 ó s_2 ó s_4.
- De manera sucesiva, sólo podrán ocurrir aquellos eventos, cuyas precedencias hayan ocurrido previamente.

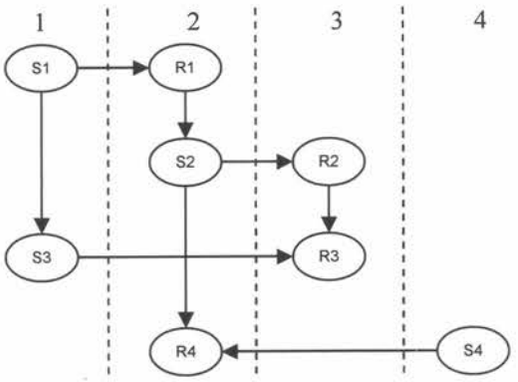


Figura 8

3.3.1.1 Definiciones

A continuación se describen definiciones importantes para el presente trabajo y que serán utilizadas a lo largo del mismo. Se define linearización completa y parcial de eventos, para interpretar secuencias de eventos válidas en el Orden Parcial de un bMSC. También se define linearización completa y parcial de etiquetas, secuencias de etiquetas que representarán las ejecuciones de los sistemas recibidos en el log de ejecución, para validar el comportamiento de los mismos.

Definición 2: Linearizaciones completas de eventos

Dado bMSC sin tiempos M y α secuencia de eventos, α es una linearización completa de eventos de M sii:

1. α es *compatible* con M.PO : α tiene los mismos eventos que el P.O. de M. (Consecuentemente pasará que $\#\alpha = \#M.PO$[3]).
2. $\alpha[i] < \alpha[j] \Rightarrow i < j \;\; \forall\, i, j : 1 \leq i, j \leq \#\alpha$: los eventos están ordenados según la relación "<" del bMSC M.

Se denomina $\mathbf{LE_c(M)}$ al conjunto de linearizaciones completas de eventos del bMSC M.

Definición 3: Linearizaciones parciales de eventos

Dado bMSC sin tiempos M, se denomina $\mathbf{LE_p(M)}$ al conjunto de linearizaciones parciales de eventos del bMSC M:

$LE_p(M) = \text{Prefijar}(LE_c(M))$

Ver definición de la operación Prefijar en el anexo, sección 11.1.1 Prefijos y Prefijar.

Observación: todo P.O. finito tiene una linearización posible.

Dado un bMSC sin tiempos $M = (I, E, g, f, \Sigma, l, \{<_i\})$, recordar que l es una función total que etiqueta cada evento de E con un símbolo del alfabeto Σ y que un bMSC especifica un Orden Parcial de los eventos en E. Si se consideran todas las posibles linearizaciones del Orden Parcial, y se mapea dicho ordenamiento a una cadena sobre Σ reemplazando cada evento e por su etiqueta asociada $l(e)$, el conjuntos de cadenas es llamado el lenguaje de M (se denota L(M)), y representa las posibles ejecuciones válidas de un sistema.

Se define la función ***label**** aplicable a una secuencia de eventos para convertir la misma en una secuencia de etiquetas.

***label**:* seq(evento) $\rightarrow$ seq(etiqueta)
$\text{label*}(\alpha) = \sigma$, donde $\forall\, \alpha[i]: l(\alpha[i]) = \sigma[i]$

Se convierte una linearización de eventos en una linearización de etiquetas a través de la función label*.

[3] #: Se utilizará el símbolo cardinal para referenciar a la cantidad de elementos de cualquier representación (secuencia, conjunto, etc).

Definición 4: Linearizaciones completas de etiquetas

Lin_c : bMSC → conj[seq(etiqueta)]

Dado el bMSC M = (I, E, g, f, Σ, l, $\{<_i\}$) : se denomina **Lin_c(M)** al conjunto de linearizaciones completas de etiquetas del bMSC M:

Lin_c (M) = { label*(α) | α: seq(E) ∧ α ∈ LE_c(M) }

Comentarios:

- Dado un bMSC sin tiempos M = (I, E, g, f, Σ, l, $\{<_i\}$), Lin_c(M) representa un conjunto de secuencias de etiquetas, que provienen de linearizaciones de eventos de M.
- Una linearización completa o parcial de eventos de un bMSC M es compatible con M.PO, si el orden dado por la secuencia de los mismos cumple con el orden determinado por M.PO. Si es completa, además tiene exactamente los mismos eventos de M.PO.

Definición 5: Linearizaciones parciales de etiquetas

Lin_p : bMSC → conj[seq(etiqueta)]

Dado el bMSC M = (I, E, g, f, Σ, l, $\{<_i\}$) : se denomina **Lin_p(M)** al conjunto de linearizaciones parciales de etiquetas del bMSC M:

Lin_p (M) = Prefijar(Lin_c(M))

Comentarios:

- Dado un bMSC sin tiempos M = (I, E, g, f, Σ, l, $\{<_i\}$), Lin_p(M) representa un conjunto de todas aquellas secuencias que cumplan ser un prefijo de cualquier linearización completa.
- La linearización parcial de M es la clausura de los prefijos de las diferentes linearizaciones completas de M.
- Ver definición de la operación Prefijar en el anexo, sección 11.1.1 Prefijos y Prefijar.

Con las definiciones anteriores se procede a enunciar y demostrar el Lema 1, posibilitando la conexión de las linearizaciones de eventos de un bMSC, con las linearizaciones de eventos de una configuración del mismo.

Lema 1: LE_p(M) = ∪ [LE_c(M') | M' es el bMSC denotado por una configuración de M]

Demostración:

⊆) $LE_p(M) \subseteq \cup [LE_c(M')$ | M' es el bMSC denotado por una configuración de M]

Sea αα' seq(E) : αα' ∈ LE_c(M) ⇒ α ∈ LE_p(M) por ser prefijo de αα'.
Se quiere ver que el conjunto de eventos de α es una configuración.
Sea a ∈ α, b ∈ M tal que b < a (según la relación "<" del bMSC).
Como a, b ∈ M

- ∃ i nat : i < #αα' tal que αα'[i] = b
- ∃ k nat : k < #αα' tal que αα'[k] = a

Y como b < a, por (2) de la definición de LE_c, **i < k**.
Como k < #α porque a ∈ α ⇒ i < #α ⇒ b ∈ α ya que el índice i < k en la secuencia α.
Por lo tanto, el conjunto de eventos de α es una configuración M' de M y α ∈ LE_c(M').

Entonces vale $LE_p(M) \subseteq \cup [LE_c(M')$ | M' es el bMSC denotado por una configuración de M].

⊇) $LE_p(M) \supseteq \cup [LE_c(M')$ | M' es el bMSC denotado por una configuración de M]

Sean α, α', α'' : seq(E),
se quiere ver que α' ∈ LE_c(M') es prefijo de α ∈ LE_c(M), es decir que ∃ α'': α = α'α'' ∈ LE_c(M).

α'' es una linearización del orden parcial de M'' = M – M', donde M'' es el bMSC que denota el orden parcial restringido al conjunto a M – M'.

Se prueba que $\alpha'\alpha''$ pertenece a la linearización de eventos de M.
Para ello se necesita probar dos items:

1. Que $\alpha'\alpha''$ es compatible con el M.PO: $\alpha'\alpha''$ cubre todos los eventos porque α' cubre M', α'' cubre M'' y $M' \cap M'' = \emptyset$.
2. Que los eventos están ordenados según la relación "<" de M: dados dos elementos cualesquiera $\alpha'\alpha''[i]$, $\alpha'\alpha''[j]$ tales que $\alpha'\alpha''[i] < \alpha'\alpha''[j]$, se quiere probar que $i < j$:
   a. Si $i, j < \#\alpha' \Rightarrow$ se debe ver que $\alpha'[i] < \alpha'[j] \Rightarrow i < j$ porque $\alpha' \in LE_c(M')$.
   b. Si $i, j \geq \#\alpha' \Rightarrow$ se debe ver que $\alpha''[i] < \alpha''[j] \Rightarrow i < j$ porque $\alpha'' \in LE_c(M'')$.
   c. Si $i < \#\alpha'$ y $j \geq \#\alpha' \Rightarrow$ se debe ver que $\alpha'[i] < \alpha''[j] \Rightarrow i < j$ sólo por el hecho de estar concatenados $\alpha'\alpha''$.
   d. Si $i \geq \#\alpha'$ y $j < \#\alpha' \Rightarrow$ se debe ver que $\alpha''[i] < \alpha'[j]$ y esto no puede pasar porque M' es una configuración y por definición es un conjunto cerrado hacia atrás, por lo tanto si $\alpha'[j] \in LE_c(M')$ entonces $\alpha'[i]$ debería también $\in LE_c(M')$.

Entonces vale $LE_p(M) \supseteq \cup [LE_c(M')$ | M' es el bMSC denotado por una configuración de M].

Como vale $LE_p(M) \subseteq \cup [LE_c(M')$ | M' es el bMSC denotado por una configuración de M].
Y también vale $LE_p(M) \supseteq \cup [LE_c(M')$ | M' es el bMSC denotado por una configuración de M].
Entonces $LE_p(M) = \cup [LE_c(M')$ | M' es el bMSC denotado por una configuración de M] y queda demostrado el lema.

El Lema 2 es análogo al Lema 1, aplicando linearización parcial y completa de etiquetas.

Lema 2: $Lin_p(M) = \cup$ [$Lin_c(M')$ | M' es el bMSC denotado por una configuración de M]

En el lema 1 se probó que $LE_p(M) = \cup [LE_c(M')$ | M' es el bMSC denotado por una configuración de M], entonces aplicando a ambos lados de la igualdad la operación label*() sobre el conjunto de eventos, la igualdad se sigue manteniendo y queda demostrado el lema.

3.3.2 Semántica de bMSC

El análisis de tiempo consiste en la validación de asignación de tiempo y en la verificación de consistencia de tiempo. Una asignación de tiempo es esencialmente una función timestamp que asocia a los eventos de un bMSC su tiempo de ocurrencia. Una asignación de tiempo es válida si respeta las restricciones de tiempo del bMSC. Un bMSC es consistente en tiempo si existe al menos una asignación de tiempo válida, que permite al bMSC tener un comportamiento donde sus eventos ocurren de acuerdo a las restricciones de tiempo especificadas.

A continuación se define secuencia de etiquetas temporizada y secuencia de eventos temporizada, es decir una secuencia de etiquetas y eventos respectivamente con asignación de tiempo.

Secuencia de Etiquetas Temporizada: Se define una Secuencia de Etiquetas Temporizada $\sigma = (l_0, ts_0)\ (l_1, ts_1)\ldots(l_m, ts_m)$ para representar un comportamiento de un bMSC, donde l_i es una etiqueta y ts_i es un número real no negativo para cualquier ($0 \leq i \leq m$) que describe el tiempo en que ocurre la etiqueta l_i. Se denomina a la etiqueta temporizada como EtiquetaTS.

Secuencia de Eventos Temporizada: Se define una Secuencia de Eventos Temporizada $\alpha = (e_0, ts_0)\ (e_1, ts_1)\ldots(e_n, ts_n)$ para representar un comportamiento de un bMSC, donde e_i es un evento y ts_i es un número real no negativo para cualquier ($0 \leq i \leq n$) que describe el tiempo en que ocurre el evento e_i. Se denomina al evento temporizado como EventoTS.

Se definen las funciones ***labelTime*** y ***labelTime****, aplicables a evento temporizado (EventoTS) y secuencia de evento temporizado (seq(EventoTS)) respectivamente; que serán de gran utilidad para la interpretación de un comportamiento válido de una secuencia de etiquetas temporizada (seq(EtiquetaTS)) para una configuración de un bMSC. Las mismas son una redefinición de la función ***l*** y ***label**** descriptas en la sección 3.3.1 Semántica de bMSC sin tiempos.

labelTime: EventoTS $\rightarrow$ EtiquetaTS $\cup \{\lambda\}$
$labelTime(e_i, ts_i) =$

si $tipoEv(e_i) = S \vee tipoEv(e_i) = R$	$(l(e_i), ts_i)$
si $tipoEv(e_i) = \theta$	λ

Convierte un evento temporizado en una etiqueta temporizada si el tipo de evento es de envío (S) o recepción (R); o se convierte en λ si el tipo de evento es de Timer (θ), porque los eventos de Timer no son visibles en la ejecución de un

sistema. Es decir que los eventos de Timer no se reciben en el log de ejecución y por lo tanto debe tenerse especial cuidado para que los mismos sean tenidos en cuenta y verificadas sus restricciones de tiempo que ejerzan en el bMSC.

labelTime*: seq(EventoTS) $\rightarrow$ seq(EtiquetaTS)
labelTime*(α) = σ, aplica la función labelTime a cada elemento $(e_i, t_i) \in \alpha$.

Convierte una linearización de eventos temporizados en una linearización de etiquetas temporizadas a través de la función labelTime*.

3.3.2.1 Definiciones

La Definición 2: Linearizaciones completas de eventos definida para un bMSC sin tiempos, también es aplicable a un bMSC, donde la secuencia de eventos recibida incluye eventos de Timer.

Definición 6: $Comp_t(\sigma, C, M_t)$ Comportamiento válido para una configuración de un bMSC

Una Secuencia de Etiquetas Temporizada $\sigma = (l_0, ts_0)\ (l_1, ts_1) \ldots (l_m, ts_m)$ representa un comportamiento válido para la configuración C de un bMSC $M_t = (I, E, g, f, \Sigma, l, \{<_i\}, T, D)$, $Comp_t(\sigma, C, M_t)$ sii:

$\exists\ \alpha$: seq(EventoTS) | $\alpha = (e_0, ts_0)\ (e_1, ts_1) \ldots (e_{m'}, ts_{m'})$ tal que

1. labelTime*(α) = σ
2. $(e_0, e_1, \ldots e_{m'}) \in LE_c(M_t')$ donde M_t' es el bMSC denotado por la configuración C
3. $\forall$ i : instancia, α_i : seq(EventoTS) proyectada a la instancia i cumple:
   3.1. $\forall\ u, v \in N : 0 \leq u \leq v \leq m' : ts_u \leq ts_v$

   3.2. $\forall\ j : 0 \leq j < \#\alpha_i : \alpha_i[j] = (e_j, ts_j) \wedge tipoEv(e_j) = \theta \wedge OpTimer(e_j) = Set$:

   Timers cerrados: si $(\exists\ k : j \leq k < \#\alpha_i : \alpha_i[k] = (e_k, ts_k) \mid tipoEv(e_k) = \theta \wedge GetTimer(e_j) = GetTimer(e_k) \wedge (\neg\exists\ (e_r, ts_r) \in \alpha : j < r < k : tipoEv(e_r) = \theta \wedge GetTimer(e_r) = GetTimer(e_j))) \Rightarrow$

   $$\boxed{\begin{array}{ll} \mathbf{OpTimer(e_k) = Reset \vee Set} & \Rightarrow \mathbf{ts_k - ts_j \leq Dur(GetTimer(e_k))} \\ \mathbf{OpTimer(e_k) = Timeout} & \Rightarrow \mathbf{ts_k - ts_j = Dur(GetTimer(e_k))} \end{array}}$$

   Timers abiertos: si $(\neg\exists\ k : j \leq k < \#\alpha_i : \alpha_i[k] = (e_k, ts_k) \mid tipoEv(e_k) = \theta \wedge GetTimer(e_j) = GetTimer(e_k)) \Rightarrow$

   $$\boxed{\mathbf{(\forall\ r : j \leq r < \#\alpha_i \mid \alpha_i[r] = (e_r, ts_r)) \Rightarrow ts_r - ts_j \leq Dur(GetTimer(e_j))}}$$

   3.3. Delay Interval de velocidad de procesamiento: $(\forall\ k, j : 0 \leq k \leq j < \#\alpha_i \mid (\exists$ Delay Interval d : ParEvDI(d) = $(\alpha_i[k], \alpha_i[j]))) \Rightarrow$

   $$\boxed{\begin{array}{c} \mathbf{IntDI(d) = [a,b] \Rightarrow a \leq ts_k - ts_j \leq b} \\ \mathbf{IntDI(d) = (a,b] \Rightarrow a < ts_k - ts_j \leq b} \\ \mathbf{IntDI(d) = [a,b) \Rightarrow a \leq ts_k - ts_j < b} \\ \mathbf{IntDI(d) = (a,b) \Rightarrow a < ts_k - ts_j < b} \end{array}}$$

4. Delay Interval de mensaje: $(\forall\ k, j : 0 \leq k \leq j < \#\alpha \mid (\exists$ Delay Interval d : ParEvDI(d) = $(\alpha[k], \alpha[j]) \wedge$ TipoDI = "ej")) $\Rightarrow$

   $$\boxed{\begin{array}{c} \mathbf{IntDI(d) = [a,b] \Rightarrow a \leq ts_k - ts_j \leq b} \\ \mathbf{IntDI(d) = (a,b] \Rightarrow a < ts_k - ts_j \leq b} \\ \mathbf{IntDI(d) = [a,b) \Rightarrow a \leq ts_k - ts_j < b} \\ \mathbf{IntDI(d) = (a,b) \Rightarrow a < ts_k - ts_j < b} \end{array}}$$

5. $(\forall\ k, j : 0 \leq k \leq j < \#\alpha : tipoEv(\alpha[k]) = S \wedge \alpha[j] = f(\alpha[k])) \Rightarrow$

   $$\boxed{\mathbf{ts_j - ts_k \geq 0}}$$

Comentarios:
Una secuencia de etiquetas temporizada σ es un comportamiento, si existe una secuencia de eventos temporizada α tal que:

1. Al aplicar labelTime*(α) se obtiene la secuencia de etiquetas temporizada σ.

2. La secuencia de eventos formada por los eventos de α es una linearización completa de eventos del bMSC que denota la configuración C.
3. Para cada instancia del bMSC, la secuencia de eventos formada por aquellos eventos que pertenecen a la instancia (proyección):
    a. Los tiempos de eventos sucesivos son crecientes.
    b. Se satisfacen todas las restricciones de tiempo establecidas por los Timers que denota la configuración parcial de M_t. **Observación:** Cuando existe algún Timer abierto t en alguna instancia y aparece un nuevo Set de t, éste actúa como un Reset + Set. Esta interpretación, compatible con la recomendación Z.120, permite que sea siempre el último Set el que se cuenta como restricción de tiempo.
    c. Se satisfacen todas las restricciones de tiempo establecidas por los Delay Intervals de velocidad de procesamiento que denota la configuración parcial de M_t.
4. Se satisfacen todas las restricciones de tiempo establecidas por los Delay Intervals de mensaje que denota la configuración parcial de M_t.
5. Se satisface la restricción de tiempo de todo mensaje, donde el tiempo del evento correspondiente a la recepción del mismo, debe ser mayor al tiempo del evento del envío.

El análisis semántico realizado se define sobre aquellos Timers para los cuales existan eventos de Set. Es decir que si existiecen eventos de Timeout o Reset de algún Timer sin sus eventos de Set, serán ignorados por la semántica.

Se redefinen la linearización completa y parcial de etiquetas aplicable ahora a un bMSC.

Definición 7: Linearizaciones completas de etiquetas temporizadas

Lin_c : bMSC $\rightarrow$ conj[seq(EtiquetaTS)]

Dado el bMSC $M_t = (I, E, g, f, \Sigma, l, \{<_i\}, T, D)$: se denomina $\mathbf{Lin_c(M)}$ al conjunto de linearizaciones completas de etiquetas temporizadas del bMSC M:

$Lin_c\ (M_t) = \{\ \sigma \mid Comp_t(\sigma, C, M_t)$ donde $C = (E, <)$ es la configuración completa de $M_t\ \}$

Comentarios:

- Dado un bMSC M_t, $Lin_c(M_t)$ representa un conjunto de secuencias de etiqueta temporizadas, tales que cada una de ellas representa un comportamiento de la configuración completa de M_t.
- C es la configuración que posee como elementos al conjunto E de M_t, es decir que posee al conjunto completo de eventos de M_t. Notar que el bMSC denotado por C es M_t.

Definición 8: Linearización parcial de etiquetas temporizadas

Lin_p : bMSC $\rightarrow$ conj[seq(EtiquetaTS)]
Dado el bMSC M_t : se denomina $\mathbf{Lin_p(M)}$ al conjunto de linearizaciones parciales de etiquetas temporizadas del bMSC M:

$Lin_p\ (M_t) = \cup\ [\ Lin_c(M_t') \mid M_t'$: bMSC denotado por una configuración parcial de $M_t\ ]$

Comentarios:

- Dado un bMSC M_t, $Lin_p(M_t)$ representa un conjunto de secuencias de EtiquetaTS que sean linearización completa de cualquier bMSC denotado por una configuración parcial de M_t.

Nota:
$Lin_p(M_t) \neq$ Prefijar($Lin_c(M_t)$) porque puede ser que la extensión de una secuencia $\sigma \in Lin_p(M_t)$ no cumpla las restricciones de tiempo.

Un ejemplo puede verse en la Figura 9 con la secuencia σ = (s1, 0) (r1, 1) (s2, 1.2) (r2, 1.5), considerando que el Timer se setea en tiempo 0.

La secuencia σ es válida hasta el momento, sin embargo la misma no es prefijo de ninguna secuencia válida porque el evento s3 debería ocurrir como mínimo en tiempo 2.5 (por la restricción del DI), contradiciendo la restricción de tiempo del Timer.

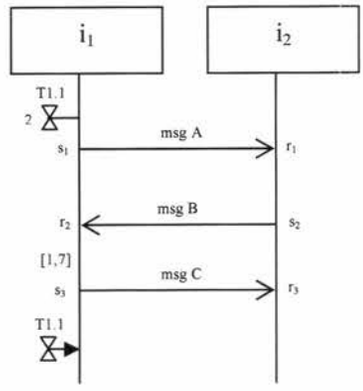


Figura 9

3.3.3 Semántica de MSC-Graph

La interpretación semántica de MSC-Graph se define a partir de la concatenación de los bMSC asociados a sus nodos.

Existen diferentes interpretaciones para la concatenación de bMSCs, sincrónica o asincrónica. Dado el ejemplo de la Figura 6, la concatenación sincrónica supone que todos los eventos de MSC 1 terminan antes de que cualquier evento de MSC 2 ocurra. La interpretación asincrónica concatena dos bMSC instancia por instancia. Para mayor detalle ver [AY99].

Se asumirá en el presente trabajo la **concatenación asincrónica** de bMSCs, por lo que se define la semántica de MSC-Graph a partir de ella.

3.3.3.1 Definiciones

Definición 9: Concatenación de dos bMSCs

Se define la operación de concatenación de dos bMSCs $\&_b$: bMSC x bMSC $\rightarrow$ bMSC.
Dado dos bMSCs M_1 y M_2 donde
$M_1 = (I_1, E_1, g_1, f_1, \Sigma_1, l_1, \{<_{1\,i}\}, T_1, D_1, tipoEv_1, OpTimer_1, GetTimer_1, Inst_1, Dur_1, IntDI_1, TipoDI_1, ParEvDI_1)$
$M_2 = (I_2, E_2, g_2, f_2, \Sigma_2, l_2, \{<_{2\,i}\}, T_2, D_2, tipoEv_2, OpTimer_2, GetTimer_2, Inst_2, Dur_2, IntDI_2, TipoDI_2, ParEvDI_2)$
la concatenación de M_1 con M_2 da como resultado el bMSC M_f donde
$M_f = (I_f, E_f, g_f, f_f, \Sigma_f, l_f, \{<_{f\,i}\}, T_f, D_f, tipoEv_f, OpTimer_f, GetTimer_f, Inst_f, Dur_f, IntDI_f, TipoDI_f, ParEvDI_f)$ definido por:

- ***Instancias***: $I_f = I_1 \cup I_2$.
- ***Eventos***: $E_f = E_1 \cup E_2$, asumiendo conjuntos de eventos disjuntos $E_1 \cap E_2 = \varnothing$ (siempre se pueden renombrar eventos para que los conjuntos sean disjuntos).
- ***Instancia de evento***: las instancias de los eventos no cambian. Para $e \in E_1$, $g_f(e) = g_1(e)$, para $e \in E_2$, $g_f(e) = g_2(e)$.
- ***Ejes***: los ejes no cambian. Para $e \in S_1$, $f_f(e) = f_1(e)$, para $e \in S_2$, $f_f(e) = f_2(e)$.
- ***Relación***: Para las instancias $i \in$ a $I_1 \setminus I_2$, $<_{f\,i} = <_{1\,i}$. Para las instancias $i \in$ a $I_2 \setminus I_1$, $<_{f\,i} = <_{2\,i}$. Para las instancias $i \in I_1 \cap I_2$, $<_{f\,i} = <_{1\,i} \cup <_{2\,i} \cup (E_{1i} \text{ x } E_{2i})$.
  (E_{1i} y E_{2i} y representan eventos de los correspondientes bMSC relacionados a la instancia i).
- ***Alfabeto***: $\Sigma_f = \Sigma_1 \cup \Sigma_2$ es el nuevo alfabeto de etiquetas.
- ***Etiquetado***: $l_f = l_1 \cup l_2$ es una función total que etiqueta cada evento de $(S_1 \cup R_1) \subseteq E_1$ con un símbolo del alfabeto Σ_1 y cada evento de $(S_2 \cup R_2) \subseteq E_2$ con un símbolo del alfabeto Σ_2, asumiendo conjunto de eventos disjuntos.
- ***Timers***: $T_f = T_1 \cup T_2$.
- ***Delay Intervals***: $D_f = D_1 \cup D_2$.
- ***tipoEv***: $tipoEv_f = tipoEv_1 \cup tipoEv_2$.
- ***OpTimer***: $OpTimer_f = OpTimer_1 \cup OpTimer_2$.
- ***GetTimer***: $GetTimer_f = GetTimer_1 \cup GetTimer_2$.
- ***Inst***: $Inst_f = Inst_1 \cup Inst_2$.
- ***Dur***: $Dur_f = Dur_1 \cup Dur_2$.
- ***IntDI***: $IntDI_f = IntDI_1 \cup IntDI_2$.
- ***TipoDI***: $TipoDI_f = TipoDI_1 \cup TipoDI_2$.
- ***ParEvDI***: $ParEvDI_f = ParEvDI_1 \cup ParEvDI_2$.

Esta definición también es aplicable en forma análoga a dos bMSCs sin tiempos, para la obtención de un nuevo bMSC sin tiempos, eliminando las operaciones sobre Timers y Delay Intervals.

Nota: Por la definición de la sección 3.2.3 Sintaxis de un MSC-Graph, las instancias son las mismas para cada nodo del MSC-Graph, por lo tanto se cumplirá siempre que toda instancia $i \in I_1 \cap I_2$.

Como ejemplo de concatenación de dos bMSCs, se utiliza dos de los bMSCs correspondientes al caso del ATM, el bMSC *ProcessPin* y el bMSC *GetOption*.

ProcessPin = $(I, E, g, f, \Sigma, l, \{<_i\}, T, D)$
I = {User, ATM, Bank}
E = $S \cup R \cup \theta = \{s_1, s_2, s_3\} \cup \{r_1, r_2, r_3\} \cup \{t_1\}$
tipoEv = $\text{tipoEv}(s_1) = \text{tipoEv}(s_2) = \text{tipoEv}(s_3) = S$ $\quad \text{tipoEv}(r_1) = \text{tipoEv}(r_2) = \text{tipoEv}(r_3) = R$ $\quad \text{tipoEv}(t_1) = \theta$
OpTimer = $\text{OpTimer}(t_1)$ = Set
GetTimer = $\text{GetTimer}(t_1)$ = T1
g = $g(s_1)$ = User; $g(r_1)$ = ATM; $g(s_2)$ = ATM; $g(r_2)$ = Bank; $g(s_3)$ = ATM; $g(r_3)$ = User; $g(t_1)$ = ATM
E_{User} = $\{s_1, r_3\}$; $E_{ATM} = \{r_1, t_1, s_2, s_3\}$; $E_{Bank} = \{r_2\}$
f = $f(s_1) = r_1$; $f(s_2) = r_2$; $f(s_3) = r_3$
f^{-1} = $f^{-1}(r_1) = s_1$; $f^{-1}(r_2) = s_2$; $f^{-1}(r_3) = s_3$
Σ = {S(ENTER_PIN), R(ENTER_PIN), S(VERIFY), R(VERIFY), S(PROCESSING), R(PROCESSING)}
l = $l(s_1)$ = S(ENTER_PIN); $l(r_1)$ = R(ENTER_PIN); $l(s_2)$ = S(VERIFY); $l(r_2)$ = R(VERIFY);
$l(s_3)$ = S(PROCESSING); $l(r_3)$ = R(PROCESSING);
$<_{User}$ = $\{s_1 < r_3\}$; $<_{ATM} = \{r_1 < t_1 < s_2 < s_3\}$; $<_{Bank} = \{\ \}$
$<$ = $[<_{User} \cup <_{ATM} \cup <_{Bank} \cup \{(s_1, r_1); (s_2, r_2); (s_3, r_3)\}]^*$
= $[\{s_1 < r_3\} \cup \{r_1 < t_1 < s_2 < s_3\} \cup \{s_1 < r_1, s_2 < r_2, s_3 < r_3\}]^*$
T = {T1}
Inst = Inst(T1) = ATM
Dur = Dur(T1) = 10
D = $\{di_1, di_2\}$
IntDI = $\text{IntDI}(di_1) = [0, 2)$, $\text{IntDI}(di_2) = [0, 2)$
TipoDI = $\text{TipoDI}(di_1)$ = "ej", $\text{TipoDI}(di_2)$ = "ej"
ParEvDI = $\text{ParEvDI}(di_1) = (s_1, r_1)$, $\text{ParEvDI}(di_2) = (s_3, r_3)$

GetOption = $(I, E, g, f, \Sigma, l, \{<_i\}, T, D)$
I = {User, ATM, Bank}
E = $S \cup R \cup \theta = \{s_4, s_5\} \cup \{r_4, r_5\} \cup \{t_2\}$
tipoEv = $\text{tipoEv}(s_4) = \text{tipoEv}(s_5) = S$ $\quad \text{tipoEv}(r_4) = \text{tipoEv}(r_5) = R$ $\quad \text{tipoEv}(t_2) = \theta$
OpTimer = $\text{OpTimer}(t_2)$ = Reset
GetTimer = $\text{GetTimer}(t_2)$ = T1
g = $g(s_4)$ = Bank; $g(r_4)$ = ATM; $g(s_5)$ = ATM; $g(r_5)$ = User
E_{User} = $\{r_5\}$; $E_{ATM} = \{r_4, t_2, s_5\}$; $E_{Bank} = \{s_4\}$
f = $f(s_4) = r_4$; $f(s_5) = r_5$
f^{-1} = $f^{-1}(r_4) = s_4$; $f^{-1}(r_5) = s_5$
Σ = {S(VALID), R(VALID), S(OPTION), R(OPTION)}
l = $l(s_4)$ = S(VALID); $l(r_4)$ = R(VALID); $l(s_5)$ = S(OPTION); $l(r_5)$ = R(OPTION)
$<_{User}$ = $\{\ \}$; $<_{ATM} = \{r_4 < t_2 < s_5\}$; $<_{Bank} = \{\ \}$
$<$ = $[<_{User} \cup <_{ATM} \cup <_{Bank} \cup \{(s_4, r_4); (s_5, r_5)\}]^* = [\{r_4 < t_2 < s_5\} \cup \{s_4 < r_4, s_5 < r_5\}]^*$
T = {T1}
Inst = Inst(T1) = ATM
Dur = Dur(T1) = 10
D = $\{di_1\}$
IntDI = $\text{IntDI}(di_1) = [0, 2)$
TipoDI = $\text{TipoDI}(di_1)$ = "ej"
ParEvDI = $\text{ParEvDI}(di_1) = (s_5, r_5)$

ProcessPin $\&_b$ *GetOption* = *(I, E, g, f, Σ, l, {<$_i$}, T, D)*

I = {User, ATM, Bank}

E = $S \cup R \cup \theta$ = $\{s_1, s_2, s_3, s_4, s_5\} \cup \{r_1, r_2, r_3, r_4, r_5\} \cup \{t_1, t_2\}$

tipoEv = $tipoEv(s_1) = tipoEv(s_2) = tipoEv(s_3) = tipoEv(s_4) = tipoEv(s_5) = S$
$tipoEv(r_1) = tipoEv(r_2) = tipoEv(r_3) = tipoEv(r_4) = tipoEv(r_5) = R$ $tipoEv(t_1) = tipoEv(t_2) = \theta$

OpTimer = $OpTimer(t_1)$ = Set, $OpTimer(t_2)$ = Reset

GetTimer = $GetTimer(t_1)$ = T1, $GetTimer(t_2)$ = T1

g = $g(s_1)$ = User; $g(r_1)$ = ATM; $g(s_2)$ = ATM; $g(r_2)$ = Bank; $g(s_3)$ = ATM; $g(r_3)$ = User; $g(t_1)$ = ATM;
$g(s_4)$ = Bank; $g(r_4)$ = ATM; $g(s_5)$ = ATM; $g(r_5)$ = User

E_{User} = $\{s_1, r_3, r_5\}$; $E_{ATM} = \{r_1, t_1, s_2, s_3, r_4, t_2, s_5\}$; $E_{Bank} = \{r_2, s_4\}$

f = $f(s_1) = r_1$; $f(s_2) = r_2$; $f(s_3) = r_3$, $f(s_4) = r_4$; $f(s_5) = r_5$

f^1 = $f^1(r_1) = s_1$; $f^1(r_2) = s_2$; $f^1(r_3) = s_3$, $f^1(r_4) = s_4$; $f^1(r_5) = s_5$

Σ = {S(ENTER_PIN), R(ENTER_PIN), S(VERIFY), R(VERIFY), S(PROCESSING), R(PROCESSING), S(VALID), R(VALID), S(OPTION), R(OPTION)}

l = $l(s_1)$ = S(ENTER_PIN); $l(r_1)$ = R(ENTER_PIN); $l(s_2)$ = S(VERIFY); $l(r_2)$ = R(VERIFY);
$l(s_3)$ = S(PROCESSING); $l(r_3)$ = R(PROCESSING); $l(s_4)$ = S(VALID); $l(r_4)$ = R(VALID);
$l(s_5)$ = S(OPTION); $l(r_5)$ = R(OPTION)

$<_{User}$ = $\{s_1 < r_3 < r_5\}$; $<_{ATM} = \{r_1 < t_1 < s_2 < s_3 < r_4 < t_2 < s_5\}$; $<_{Bank} = \{r_2 < s_4\}$

< = $[<_{User} \cup <_{ATM} \cup <_{Bank} \cup \{(s_1, r_1); (s_2, r_2); (s_3, r_3); (s_4, r_4); (s_5, r_5)\}]^*$
= $[\{s_1 < r_3 < r_5\} \cup \{r_1 < t_1 < s_2 < s_3 < r_4 < t_2 < s_5\} \cup \{r_2 < s_4\} \cup \{s_1 < r_1, s_2 < r_2, s_3 < r_3, s_4 < r_4, s_5 < r_5\}]^*$

T = {T1}

Inst = Inst(T1) = ATM

Dur = Dur(T1) = 10

D = $\{di_1, di_2, di_3\}$

IntDI = $IntDI(di_1) = [0, 2)$, $IntDI(di_2) = [0, 2)$, $IntDI(di_3) = [0, 2)$

TipoDI = $TipoDI(di_1)$ = "ej", $TipoDI(di_2)$ = "ej", $TipoDI(di_3)$ = "ej"

ParEvDI = $ParEvDI(di_1) = (s_1, r_1)$, $ParEvDI(di_2) = (s_3, r_3)$, $ParEvDI(di_3) = (s_5, r_5)$

Definición 10: Nodos consecutivos en un MSC-Graph

Dado un MSC-Graph G = (B, V^I, N, suc, getbMSC); $v_i, v_j \in V^I \cup N$ se dicen *Nodos consecutivos* en G sii ($\exists$ ej $\in$ suc : ej = (v_i, v_j)).

Definición 11: Camino en un MSC-Graph

Dado un MSC-Graph G = (B, V^I, N, suc, getbMSC), un *Camino* en G es una secuencia de *Nodos consecutivos* $v_0\ v_1 \ldots v_n$ en G (Ver Definición 10) tal que $v_i \in V^I \cup N$.

Definición 12: Camino de aceptación

Dado un MSC-Graph G = (B, V^I, N, suc, getbMSC), un *Camino de aceptación* es un *Camino* $v_0 \ldots v_n$ en G con $v_0 = V^I$.

Definición 13: bMSC de un camino de aceptación

Dado un *Camino de aceptación* ρ, M_ρ = getbMSC(v_0) $\&_b$ getbMSC(v_1) $\&_b \ldots \&_b$ getbMSC(v_n) es el bMSC formado por la concatenación de los bMSCs de cada $v_i \in \rho$.

Definición 14: Lenguaje asincrónico de un MSC-Graph

El lenguaje asincrónico de un MSC-Graph G, denotado $L^a(G)$ es el conjunto $\{Lin_c(M_\rho) \mid \rho$ es un Camino de aceptación en G$\}$. Se asocia un bMSC con cada camino, concatenando **asincrónicamente** bMSCs correspondientes a vértices individuales. El lenguaje del MSC-Graph es la **unión** de los lenguajes de los bMSCs asociados con los Caminos de aceptación.

3.4 Sintaxis Concreta

Para representar la sintaxis concreta de bMSCs y MSC-Graphs, se decidió utilizar la gramática textual definida por la recomendación Z.120 para el intercambio de MSCs entre diferentes herramientas. Esta gramática ofrece dos formas principales para describir bMSCs:

1. **Orientada a instancia:** un bMSC se describe con la sucesión de los comportamientos de cada una de sus instancias en forma aislada.
2. **Orientada a evento:** un bMSC se describe con una lista de eventos, por ejemplo como son experados que ocurran en un seguimiento del sistema o como son encontrados mientras es recorrido el bMSC de principio a fin.

La sintaxis propuesta es una porción de la gramática textual orientada a evento recomendada por Z.120, porque se redujo al alcance y definiciones descriptos con anterioridad en el presente trabajo.

3.4.1 Introducción a la sintaxis concreta de bMSC

La gramática textual de bMSC consiste en las palabras claves **msc** y **endmsc**, y entre ellas un *msc name* y un *msc body*. En la sintaxis orientada a evento, un *msc body* consiste en una lista de definiciones de eventos. Una definición de evento es un *instance name* seguido de un *instance event*.

Los *instance event* son *message events*. Un *message event* se describe por un *message output* y un *message input*. Si ***m*** es un mensaje que es enviado desde la instancia ***i*** a la instancia ***j***, en forma textual el evento de *message output* se denota "*i:* ***out*** *m* ***to*** *j*" y el evento de *message input* "*j:* ***in*** *m* ***from*** *i*".

Un evento de *message output* y un evento de *message input* se corresponden sii:

1. Los eventos tienen el mismo nombre de mensaje.
2. La instancia en la cual el evento de *message output* es especificado, es la misma que la instancia indicada por la dirección de recepción del evento de *message input*.
3. La instancia en la cual el evento de *message input* es especificado, es la misma que la instancia indicada por la dirección de envío del evento de *message output*.

Un requerimiento natural de la gramática textual recomendada por Z.120 es que para cada evento de *message output* hay uno y sólo un evento de *message input*.

En la sección 11.2 del Anexo A se define la gramática textual que representa la sintaxis concreta de un bMSC. La misma se despliega en varias secciones con porciones de la gramática, que en conjunto definen la sintaxis concreta. Cada una de las secciones posee una breve descripción que ayudará a una mejor comprensión. Antes de la definición, el lector podrá encontrar una breve introducción que permitirá una correcta interpretación.

Ejemplo del bMSC de la Figura 5 descripto en forma textual orientada a evento:

```
mscdocument Figura5;
msc Figura5;
i1: instance;
i2: instance;
i3: instance;
i1: out msgA ((1,2]) to i2;
i2: in msgA ((1,2]) from i1;
i1: set T1.1 (3);
i3: set T3.1 (5);
i2: out msgB to i3;
i3: in msgB from i2;
i3: out msgC to i2;
i2: in msgC from i3;
i3: timeout T3.1;
i3: out msgE to i1;
i2: out msgD to i1;
i1: in msgD from i2;
i1: reset T1.1;
i1: in msgE from i3;
int_boundary ib1 is attached to in msgB out msgC [2,4)
endmsc;
```

3.4.2 Introducción a la sintaxis concreta de MSC-Graph

La gramática textual de MSC-Graph consiste en las palabras claves **msc** y **endmsc**, y entre ellas un *mscgraph name* y un *mscgraph body*. Un *mscgraph body* consiste en un id de nodo inicial, seguido de una lista de *node expression*. Un *node expression* consiste en un id de nodo, seguido del *bMSC name* referenciado y una lista de ids correspondientes a nodos hijos.

En la sección 11.2 del Anexo A se define la gramática textual que representa la sintaxis concreta de un MSC-Graph.

Ejemplo del MSC-Graph de la Figura 11 descripto en forma textual:

```
mscdocument Figura11;
msc Figura11;
expr L1;
 L1: (MSC1) seq (L2 alt L3);
 L2: (MSC2) seq (L4);
 L3: (MSC3) seq (L1);
 L4: end;
endmsc;
```

3.4.3 Introducción a la sintaxis concreta de log de ejecución

El log de ejecución está formado por la concatenación de pares de elementos: timestamp y envío o recepción de una etiqueta.

Su formato es el siguiente:

$Timestamp_1$ Et_1

$Timestamp_2$ Et_2

$Timestamp_3$ Et_3

...

$Timestamp_n$ Et_n

donde Et_i es la etiqueta S(x) o R(x) correspondiente al envío (S) o recepción (R) del nombre del mensaje (x), y $Timestamp_i$ al tiempo en que ocurrió la misma.

En la sección 11.2 del Anexo A se define la gramática textual que representa la sintaxis concreta de un log de ejecución.

Ejemplo:

```
2002-09-12 10:15:23.123 S(msg1)
2002-09-12 10:15:30.25633 R(msg1)
```

4 Solución algorítmica a los problemas de verificación

4.1 Notación

Se utilizará una pseudo-lógica para exponer enunciados formales de los tipos de datos y su relación con el algoritmo que resuelve el problema analizado.

Para esto, se realiza un abuso de la notación, alterando la sintaxis de los predicados de lógica por otros con notación "Orientada a Objetos", facilitando de esta forma su comprensión.

Por ejemplo, para saber si al agregar a una configuración un nuevo elemento, ésta sigue siendo una configuración; en lugar de utilizar "esConfiguracion (conf $\cup$ { e }) de bMSC", se utilizará "bMSC.esConfiguracion?(conf $\cup$ { e })".

4.2 Pasos a seguir

En las próximas secciones se muestra en forma evolutiva la construcción de las soluciones algorítmicas para resolver los problemas de monitoreo, creciendo en la complejidad y agregando en forma incremental un alcance más amplio.

Las etapas a seguir son:

1. **Monitoreo de bMSCs sin tiempos:** la solución resuelve el problema de analizar cuándo una posible ejecución de un sistema es válida de acuerdo al comportamiento especificado a nivel bMSC y sin restricciones de tiempo.

2. **Monitoreo de MSC-Graphs sin tiempos:** la solución resuelve el problema de analizar cuándo una posible ejecución de un sistema es válida de acuerdo al comportamiento especificado a nivel MSC-Graph y sin restricciones de tiempo.

3. **Monitoreo de MSC-Graphs:** a la solución propuesta anterior se le agregan las validaciones correspondientes a la verificación de las restricciones de tiempo que se encuentran en la especificación del comportamiento del sistema a nivel MSC-Graph. Además se optimizan las estructuras utilizadas.

Notar que los invariantes de los algoritmos se encuentran en las pruebas de corrección de los mismos, a continuación de ser presentados.

4.3 Monitoreo de bMSCs sin tiempos

La primera y más simple de las soluciones, resuelve el problema de analizar posibles ejecuciones de un sistema, para validar su correctitud de acuerdo al comportamiento especificado a nivel bMSC sin restricciones de tiempo.

A continuación se muestra un ejemplo de bMSC sin tiempos que se utilizará para mejorar la comprensión de los conceptos introducidos:

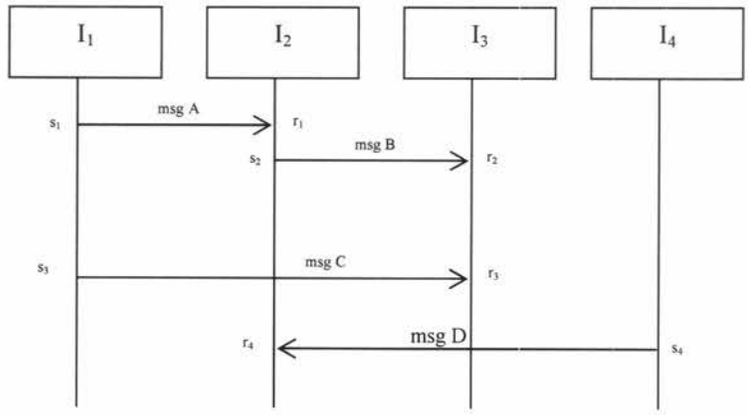


Figura 10

4.3.1 Definiciones

Definición 15: Evento Minimal

Dado un evento a, a es *Evento Minimal* sii $\neg\exists$ evento $b \mid b < a$

En el ejemplo de la Figura 10, S1 y S4 son eventos minimales.

4.3.2 Analizando un bMSC sin tiempos

A continuación se detallan los formalismos y el algoritmo asociado para realizar el monitoreo de sistemas especificados con bMSCs sin tiempos.

El tipo abstracto de datos más importante para el monitoreo de bMSCs es el TAD BMSCMonitor. Este se utiliza para llevar el seguimiento y controlar si una secuencia de etiquetas es una linearización parcial de un bMSC.

Para las Pre y Post condiciones del TAD BMSCMonitor se utilizarán las operaciones *virtuales* mBMSC, mConfig y mSeqRec. Estos proyectores devuelven respectivamente el bMSC subyacente, la configuración subyacente y la secuencia de etiquetas recorrida hasta el momento. Cuando un proyector no es utilizado en las Pre y Post condiciones de una operación, debe interpretarse que no se ve modificado.

Para utilizar los proyectores virtuales desde afuera del tipo, se utilizará el nombre del proyector sin la letra m, por ejemplo, para mBMSC se usará BMSC().

Las operaciones del TAD más importantes se describen a continuación:

TADs utilizados

tipo BMSCMonitor

operaciones más importantes

```
//Crea un nuevo BMSCMonitor a partir de un bMSC
constructor(in b: BMSC)
{True}
{mBMSC = b ∧ mConfig.esVacia?() ∧ mSeqRec.esVacia?()}

//Busca los eventos e tales que l(e) = et y que pueden extender la configuración del bMSC
//Se sabe que existirá ninguno o un evento
buscar(in et: Etiqueta)          → ret_value : Secuencia_Evento
{True}
{(∀ e ∈ ret_value : mBMSC.pertenece?(e) ∧ mBMSC.getLabel(e) = et ∧
 mBMSC.esConfiguracion?(mConfig ∪ {e}))}

//Extiende la configuración del bMSC con el evento recibido
avanzar(in ev: Evento)
{mConfig = conf0 ∧ mSeqRec = seqRec0 ∧ mBMSC.esConfiguracion?(mConfig ∪ {ev})}
{mConfig = conf0 ∪ {ev} ∧ mSeqRec = seqRec0 • mBMSC.getLabel(ev)}
```

4.3.3 Algoritmo

El algoritmo consta de una operación denominada "esPrefijo?", que se encarga de determinar si una secuencia de etiquetas es un prefijo válido del lenguaje denotado por el bMSC que especifica el comportamiento del sistema a monitorear.

```
//Determina si una secuencia de etiquetas es prefijo válido del lenguaje denotado por el bMSC subyacente
esPrefijo?(in b: bMSC, in s: Secuencia_Etiqueta)          → ret_value : Bool
{True}
{ret_value = (s ∈ Lin_p(b))}
```

```
esPrefijo?(in bmsc: bMSC,  in s: Secuencia_Etiqueta)      → ret_value : Bool
  var
      seqEv          : Secuencia_Evento
      matchea        : Bool
      bMSCMonitor    : BMSCMonitor

  begin
      matchea        := true
      bMSCMonitor    := BMSCMonitor.constructor(bmsc)

      while (not s.esVacia?() and matchea)

            seqEv := bMSCMonitor.buscar(s.prim())
            //Buscar evento[4] e que matchee con la etiqueta actual, tal que su ocurrencia extienda
            //la configuración por una linearización válida.

            if  (not seqEv.esVacia?()) then
                  bMSCMonitor.avanzar(seqEv.prim())
                  s := s.fin()
            else
                  matchea := false
            endif

      endwhile

      ret_value := matchea

  end
```

4.3.4 Prueba de corrección

Este algoritmo reconoce todas las posibles linearizaciones aceptadas por el bMSC recibido.

El algoritmo para monitorear la correcta ejecución de un bMSC, consiste en recorrer el grafo generado por el orden parcial del bMSC, utilizando un iterador que recorre la configuración actual en cada momento, para lograr avanzar por diferentes caminos, según las etiquetas recibidas.

Por cada etiqueta recibida de la secuencia de entrada, se busca el evento que matchee con ella, que pueda ocurrir a partir de la configuración actual del bMSC (denotada por *bMSCMonitor*.Conf()). Se sabe que el algoritmo es determinístico, ya que este paso devuelve a lo sumo un evento. Se puede asegurar esto gracias a la "Restricción 1: Alcance de las etiquetas de los mensajes" en la sección 3.2.1.

En este contexto habrá un único evento o ninguno que matchee con una etiqueta recibida. La instancia a la que pertenece este evento (en caso que exista un evento) es única por la definición. Entonces dada una determinada configuración parcial del bMSC, se podrá extender la misma sólo por un evento posible.

- *Si existe un evento asociado* (caso que buscar devuelva algún elemento): avanzar por el evento recibido, marcándolo como visitado y actualizando la configuración actual.
- *Si no existe un evento asociado*: se recibió una etiqueta tal que la secuencia no es válida.

[4] Esta operación es determinística, se sabe que habrá un único evento (o ninguno), tal que está asociado a una etiqueta en este contexto. Ver "Restricción 1: Alcance de las etiquetas de los mensajes".

Invariante

bMSCMonitor.Conf() es una configuración del *PO* del *bMSCMonitor*.BMSC() $\wedge$
(PI(s) $\in$ Lin_c(M') | M': bMSC denotado por la configuración *bMSCMonitor*.Conf()) $\wedge$ s_0 = PI(s) & PD(s)

Para más detalles de "es una configuración de", recordar la sección 3.1 "Conceptos teóricos".

PI(s) representa la secuencia de etiquetas analizada hasta el momento en el algoritmo, o sea PI(s) & PD(s) = s_0 (PD(s) es la secuencia de etiquetas aún no analizada).

El invariante del algoritmo asegura que:

- La configuración del Monitor es una configuración del Orden Parcial del bMSC analizado. Esto sucede por la construcción del BMSCMonitor y por el uso del TAD. El Orden Parcial se mantiene por la forma en que se utiliza el "Avanzar", ya que sólo se avanza si se mantiene el Orden Parcial.
- La secuencia de etiquetas reconocida hasta el momento es una Linearización Completa del bMSC denotado por la configuración actual.
- Se desprende del Lema 2 que la secuencia de etiquetas reconocida hasta el momento es una linearización parcial de bMSC analizado.

Si al finalizar el procesamiento de la última etiqueta recibida, la ejecución continúa activa, significa que la secuencia recibida es un prefijo válido del lenguaje denotado por el bMSC analizado.

En caso que sea exitoso, el algoritmo asegura que la secuencia es una linearización completa de etiquetas de una configuración del PO del bMSC.

Por el Lema 2 enunciado en 3.3.1, y la forma de construcción del algoritmo que en todo momento asegura que la secuencia analizada es una linearización completa de una configuración del PO del bMSC, se puede concluir que el algoritmo reconoce una linearización parcial del bMSC analizado.

4.4 Monitoreo de MSC-Graphs sin tiempos

La siguiente solución resuelve el problema de analizar posibles ejecuciones de un sistema, para validar su correctitud de acuerdo al comportamiento especificado a nivel MSC-Graph sin restricciones de tiempo.

A continuación se muestra un ejemplo de MSC-Graph sin tiempos que se utilizará para mejorar la comprensión de los conceptos introducidos:

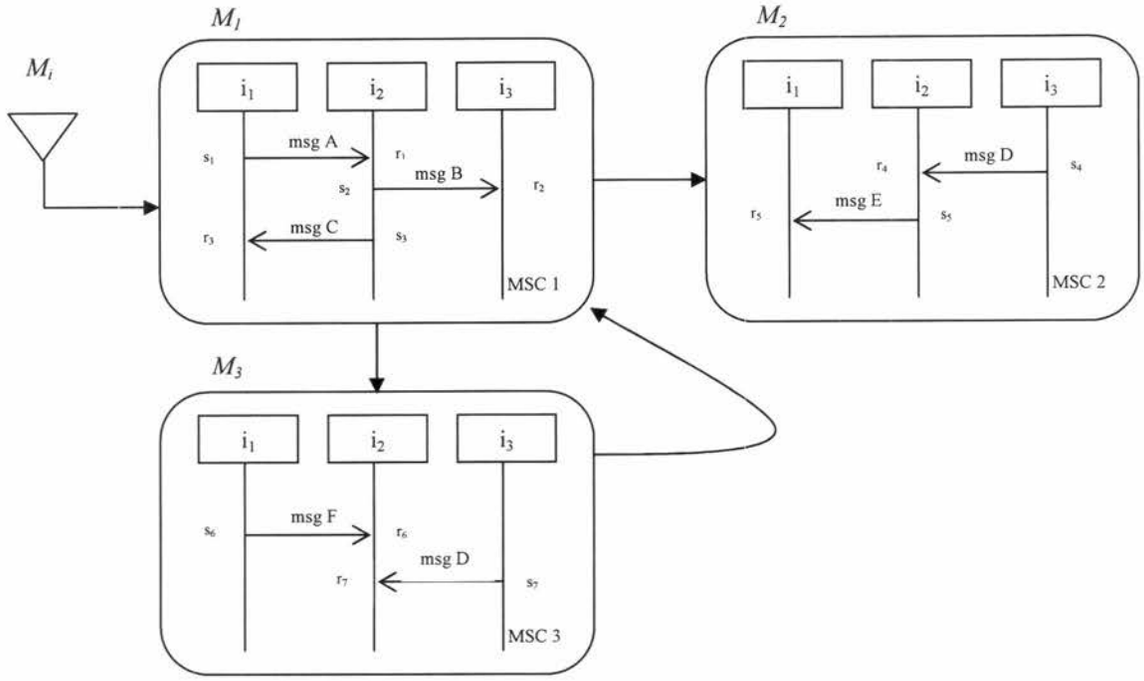


Figura 11

4.4.1 Definiciones

Definición 16: Camino testigo de una secuencia de etiquetas σ: CaTSE σ

Un *Camino testigo de una secuencia de etiquetas* σ en el MSC-Graph G, es un *Camino de aceptación*[5] C en G tal que:

1. $\sigma \in Lin_p(M_C)$
   La secuencia de etiquetas σ pertenece a la linearización parcial de la concatenación de los bMSCs de un camino C.
2. $\neg \exists$ C' Camino | C' $\subset$ C : $\sigma \in Lin_p(M_{C'})$
   No existe otro camino llamado C' donde la secuencia también pertenezca a la linearización parcial de la concatenación de los bMSCs de C'. Esto significa que el Camino testigo de una secuencia de etiquetas es minimal y por lo tanto no posee un sufijo no utilizado.

Nota: Se denomina CaTSEs al conjunto de "Caminos testigo de una secuencia de etiquetas".

Definición 17: Camino hacia una etiqueta et: CHE et

Dado un MSC-Graph *G* y una etiqueta *et*, un *Camino hacia et* es un *Camino C* en *G* tal que:

1. $(\exists\, e$: evento | $e \in M_{Ult(C)} \wedge e$ es minimal $\wedge\ l(e) = et)$
   Existe un evento en el bMSC del último nodo del camino tal que es minimal, y matchea con la etiqueta *et*.
2. $\neg\, (\exists\, e$: evento | $e \in M_{Com(C)} \wedge l(e) = et\,)$
   No existe un evento en los bMSCs de los nodos previos al último, que matchee con la etiqueta *et*.

Notas:
Com y Ult son operaciones definidas sobre una secuencia. Com representa toda la secuencia menos el último elemento y Ult el último elemento.

[5] Ver Definición 12.

Se denomina CHEs al conjunto de "Caminos hacia una etiqueta".

En el ejemplo de la Figura 11, si se está parado en el evento r_2, y se desea saber los caminos posibles para alcanzar la etiqueta S(D), se tendrán los caminos { <M2>; <M3> }.
En el ejemplo de la Figura 12, si se está parado en el evento r_2, y se desea saber los caminos posibles para alcanzar la etiqueta S(E), se tendrán los caminos { <M2–M3>; <M3> }.
En el ejemplo de la Figura 13, si se está parado en el nodo inicial, y se desea saber los caminos posibles para alcanzar la etiqueta S(A), se tendrán los caminos { <M1>; <M2> }

Los CHEs se utilizan en el algoritmo principal en el momento que se necesita hacer una búsqueda extendida, es decir, encontrar un camino desde un nodo "actual" hacia una etiqueta.

4.4.2 Analizando MSC-Graphs sin tiempos

Monitoreo sin ciclos

Si se supone un MSC-Graph donde no existen ciclos, tomando como ejemplo el MSC-Graph de la Figura 11 (no considerar la transición M3-M1), ¿cómo determinaría el lector si una secuencia de ejecución pertenece al lenguaje dado por el MSC-Graph?

Considerando la concatenación asincrónica de bMSCs como se detalla en Definición 9, un algoritmo sencillo podría generar todos los posibles Caminos de aceptación (tal que el último nodo del camino no pueda extenderse), concatenando los bMSCs correspondientes a cada nodo del camino, para recorrer todas las posibles ejecuciones válidas. Esto podría generar un no-determinismo, explicado más adelante. Según el ejemplo, se deberían concatenar M1-M2 y M1-M3. De esta manera, el problema de encontrar las ejecuciones válidas del ejemplo, se resume a analizar el monitoreo de tantos bMSCs como Caminos de aceptación existan (sin pérdida de generalidad, podría optimizarse esta técnica, pero no lo haremos aquí ya que nos interesa profundizar en otro tipo de problema).

Monitoreo con ciclos

Es necesario contemplar el análisis de situaciones de ejecución donde el no-determinismo está presente. Esto puede ocurrir cuando existen diferentes CHEs posibles, o existen diferentes eventos que se asocian a una misma etiqueta. A continuación se detallan estas situaciones:

Condiciones de no-determinismo en las transiciones

Existen ciertos casos donde la ocurrencia de una etiqueta puede generar una transición no-determinística:

A. Dos CHEs, donde uno tiene una instancia Idle
A continuación se muestra un ejemplo ilustrativo:

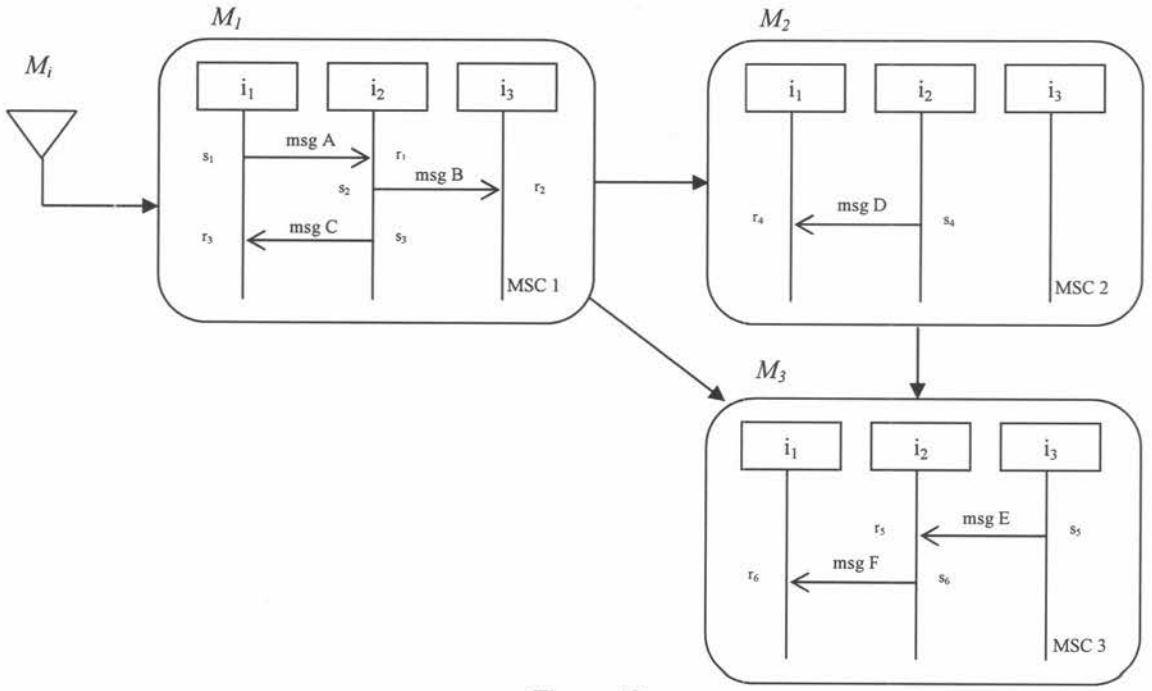


Figura 12

El problema del análisis de este MSC-Graph, sucede con la ocurrencia de la etiqueta S(E), asociada al evento s_5 en ciertas circunstancias: si se toma como ejemplo la ocurrencia de los eventos: s_1, r_1, s_2, r_2; y se analiza luego la ocurrencia del evento s_5, ¿qué se debe considerar como concatenación de bMSCs? ¿Tomar M_1-M_3, o tomar M_1-M_2 y luego el resultante con M_3? El hecho que la instancia i_3 no posea eventos en M_2 (instancia Idle) genera un no-determinismo que puede ser analizado suponiendo todos los CHEs posibles.

B. Considerando múltiples eventos que matchean hacia una misma etiqueta:
A continuación se muestra un ejemplo ilustrativo:

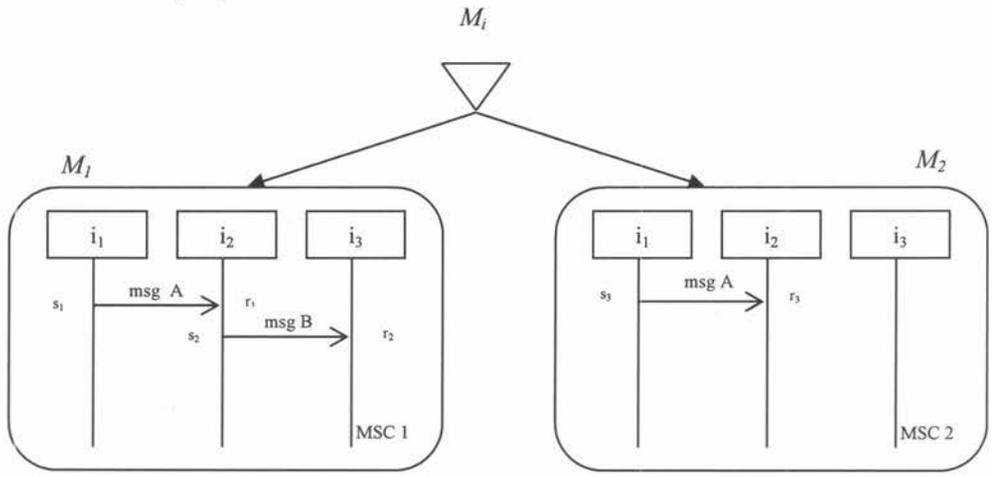


Figura 13

El problema del análisis de este MSC-Graph, sucede con la ocurrencia de la etiqueta S(A) (Send "A"). Se puede ver que tanto s_1 como s_3 se encuentran asociadas a la etiqueta S(A). Cuando se recibe en la secuencia de ejecución S(A) como primera etiqueta, es posible asociarla al bMSC M_1, así como al M_2.

También ocurre esto en el ejemplo de la Figura 11, con la etiqueta S(D) en los bMSCs M2 y M3.

Para resolver A y B, se deben controlar dos ejecuciones posibles, suponiendo CaTSEs diferentes en cada una de ellas, dado que es posible alcanzar la etiqueta en más de una forma. En esta sección se formalizarán estas ideas.

Se define a continuación una familia de MSC-Graphs, denominada MSC-Graphs con ciclos con instancia Idle, circunstancia muy conocida en la literatura y cuyo uso se restringirá en el presente trabajo.

Definición 18: Ciclo en un MSC-Graph

Sea MSC-Graph G, se define un *Ciclo* en G, como un *Camino* $v_0\ v_1\ ....\ v_n$ en G (Ver Definición 11), tal que $(\forall\ i, j : 0 \leq i, j \leq n \wedge i \neq j : v_i \neq v_j) \wedge (v_n, v_0)$ son *Nodos consecutivos* (Ver Definición 10).

Definición 19: Ciclo con instancia Idle: CID

Sea MSC-Graph G, se define un *Ciclo con instancia Idle* en G, como un *Ciclo* $v_0\ v_1\ ....\ v_n$ en G (Ver Definición 18), donde ($\exists$ in: Instancia | ($\forall\ v_j$.getbMSC() : $0 \leq j \leq n \wedge v_j$.getbMSC() = (I, E, g, f, Σ, l, $\{<_i\}$) : in $\in$ I $\wedge$ in es *Instancia Idle* en v_j.getbMSC())).

Se denomina CIDs a Ciclos con instancia Idle.

Un CID produce un problema al momento de conocer la cantidad de veces que se recorrió un ciclo antes de haber salido del mismo. Al generarse esta situación, los posibles CaTSEs que podrían reconocer secuencias válidas para el lenguaje definido por el MSC-Graph, pasan a ser infinitos.

A continuación se muestra un ejemplo:

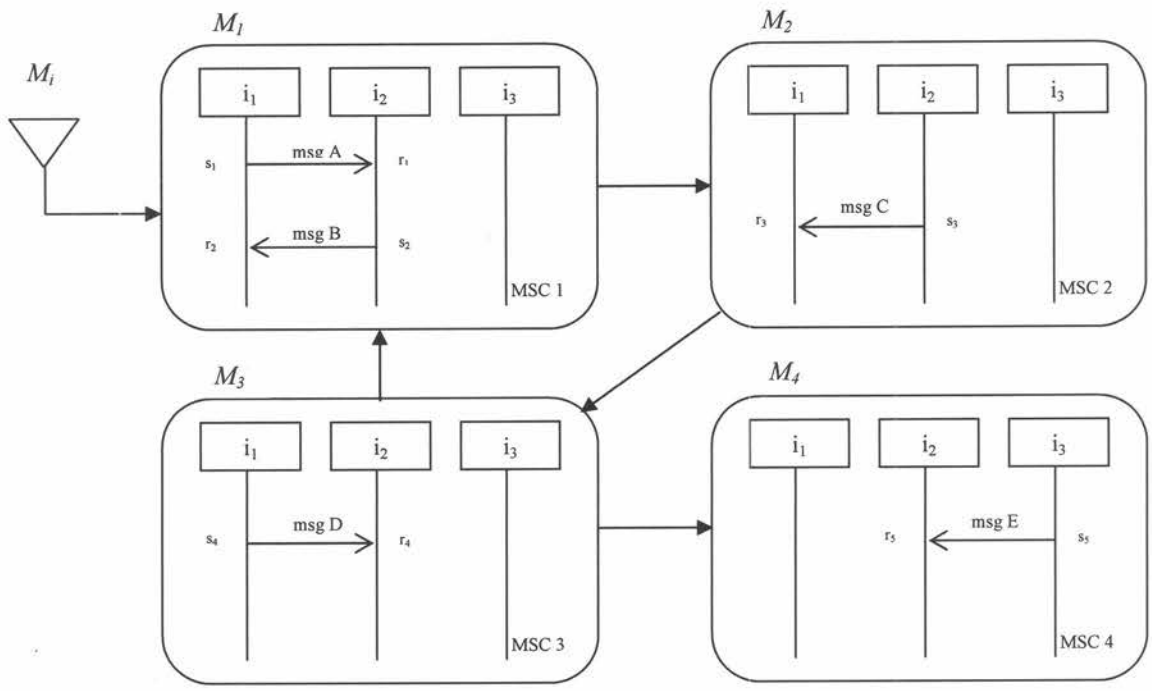


Figura 14

Se supone la secuencia de ejecución S(A), R(A), S(B), S(C) y luego la ocurrencia de S(E). Para la secuencia de ejecución se tiene un CaTSE, cuyos nodos recorridos son M_1 y M_2.

Para el CaTSE que se está analizando, se deberá realizar una búsqueda extendida, ya que la ocurrencia de la etiqueta S(E) pertenece a un bMSC asociado a un nodo no recorrido. Esta búsqueda debería obtener todos los posibles CHEs existentes desde el nodo actual (M_2) y con la configuración actual, para encontrar el evento que matchee con la etiqueta S(E) recibida.

El gran problema de este tipo de MSC-Graphs, es que la cantidad de CHEs existente es infinita. Esto se debe a que es posible realizar infinitas vueltas en cualquier CID que el MSC-Graph posea, formando distintos CHEs y por lo tanto generando infinitos CaTSEs. En el ejemplo de la Figura 14 es posible ciclar infinitamente por el CID integrado por los nodos M_1, M_2 y M_3, y luego llegar al nodo M_4 donde se encuentra un evento minimal que matchee con S(E).

En el presente trabajo, se utilizarán MSC-Graphs sin CIDs, por lo tanto el algoritmo implementado reconocerá una porción del lenguaje de un MSC-Graph con CIDs.

Teorema:
Si no existen ciclos con instancias Idle, toda secuencia de etiquetas σ (cadena del lenguaje) del MSC-Graph tiene una cantidad finita de CaTSEs σ.

Lema: Para toda secuencia de etiquetas σ, existe un $k \in \mathbb{N}$ tal que $\forall$ C CaTSE σ, $\#C \leq k$ (es decir existe una cota para la longitud de C).

Observación: de aquí se desprende que la cantidad de CaTSEs es finita ya que el branching del MSC-Graph es finito.

Demostración del lema:
Si k no existiese, existe un CaTSE que hace ejecutar algún ciclo m veces, para cualquier m que elija.
Supongamos $m > \#\sigma$, ya que un CaTSE de longitud arbitraria sobre un MSC-Graph finito necesariamente ejecuta algún ciclo de manera no acotada.

Por definición de CaTSE, su minimalidad implica que en el último bMSC del CaTSE C hay un elemento de σ, digamos $\sigma[i]$, que en la linearización se corresponde con un evento e del último bMSC de C (sino ese último bMSC está demás y por ende el CaTSE no sería minimal).

Sea *it* la instancia de $\sigma[i]$. Al ejecutar *m* veces un determinado ciclo que no tiene instancias Idle, implica que hay al menos *m* eventos de *it* que aparecen antes del evento *e* (ya que *it* no es idle). Como la linearización preserva el orden, esos eventos son matcheados por elementos de σ.

Absurdo porque $\#\sigma < m$!

La utilización de este teorema es muy importante para el análisis que se presentará en las secciones siguientes, debido a que las definiciones se basan en él por la existencia de una cantidad finita de CaTSEs. En particular, se la requiere para la definición de Testigo (se verá que es equivalente a CaTSE).

Habría que generalizar la definición para poder soportar CIDs, pero este desarrollo queda fuera del alcance del presente trabajo. Sin embargo, en la sección "8 Trabajos a futuro" se presenta un breve análisis, junto con algunas sugerencias útiles para iniciar su estudio e implementación.

TADs utilizados

A continuación se detallan los formalismos y el algoritmo asociado para realizar el monitoreo de sistemas especificados con MSC-Graphs.

El tipo abstracto de datos más importante para el monitoreo de MSC-Graphs es el TAD Testigos, que posee una secuencia del TAD Testigo. Cada Testigo posee un BMSCMonitor subyacente, que contiene a su vez un bMSC para mantener un CaTSE en el MSC-Graph analizado. Es decir, que cada Testigo representa un CaTSE en el MSC-Graph.

El TAD Testigos permite monitorear si una secuencia de etiquetas es una linearización del bMSC de algún CaTSE en el MSC-Graph analizado. Tiene la funcionalidad de mantener múltiples testigos (CaTSEs) activos, permitiendo el no-determinismo en la elección de los posibles CaTSEs a seguir. Esto implica tener múltiples BMSCMonitor en simultáneo, cada uno con diferentes bMSCs subyacentes y configuraciones diferentes.

Para las Pre y Post condiciones del TAD Testigos se utilizarán las operaciones *virtuales* PI (Parte Izquierda), PD (Parte Derecha), mTestigosAct y mMSCGraph. Las tres primeras operaciones se aplican sobre la Secuencia de testigos subyacente, devolviendo Secuencia_Testigo, y la última devuelve el MSCGraph subyacente. La operación mTestigoAct representa la Secuencia_Testigo de todos los testigos activos y está conformada por PI & PD. Cuando un proyector no es utilizado en las Pre y Post condiciones de una operación, debe interpretarse que no se ve modificado.

Para utilizar los proyectores virtuales desde afuera del tipo, se utilizará el nombre del proyector sin la letra m, por ejemplo, para mTestigosAct se usará TestigosAct().

Las operaciones del TAD más importantes se describen a continuación:

tipo Testigos

operaciones más importantes

```
//Crea un monitor con un único testigo que tiene un bMSC con el nodo inicial
//El iterador se encuentra posicionado en el único testigo existente
//Se recibe el MSC-Graph a monitorear
constructor(in mg : MSCGraph)
{True}
{mMSCGraph = mg ∧ PI = ∅ ∧ PD = (Testigo.constructor(mg)).concatCamino(mg.V^I)}

//Indica si el iterador ha terminado de iterar
hayProximo?()                    → ret_value : Bool
{True}
{ret_value = (PD ≠ ∅)}

//Devuelve el próximo Testigo según el iterador pero no modifica el estado del mismo
proximoTestigo()                 → ret_value : Testigo
{hayProximo?() ∧ PI = PI0 ∧ PD = PD0}
{ret_value = Prim(PD0) ∧ PI = PI0 ∧ PD = PD0}
```

```
//Elimina el Testigo actual de la secuencia de testigos
eliminarTestigo()
{hayProximo?() ∧ PI = PI0 ∧ PD = PD0}
{PI = PI0 ∧ PD = Resto(PD0)}

//Posiciona el iterador al comienzo
arrancarTestigos()
{PI = PI0 ∧ PD = PD0}
{PI = ∅ ∧ PD = PI0 & PD0}

//Extiende el testigo actual con nuevos CHEs hacia la etiqueta actual
//Esto agrega los nuevos testigos en la "Parte Derecha" del iterador tal que todos tienen la misma configuración
//inicial (igual a la del actual)
extenderTestigo(in secCam : Secuencia_Camino)
{hayProximo?() ∧ PI = PI0 ∧ PD = PD0 ∧
 (∀ cam : Camino | cam ∈ secCam : cam es Camino en mMSCGraph ∧
 (Prim(PD0).getNodoActual(), cam.prim()) ∈ mMSCGraph.suc()}
 {PI = PI0 ∧ PD = Map(λc • (Prim(PD0).concatCamino(c)), secCam) & Resto(PD0)}
//Se reemplaza Prim(PD0) por la ocurrencia que resulta de concatenarlo contra cada
//extensión de secCam (aplicándole la operación concatCamino)

//Avanza al próximo Testigo según el iterador
avanzarTestigo()
{hayProximo?() ∧ PI = PI0 ∧ PD = PD0}
{PI = PI0 • Prim(PD0) ∧ PD = Resto(PD0)}

//Determina si existe algún testigo activo en la secuencia subyacente
existenTestigosActivos?()          → ret_value : Bool
{True}
{ret_value  = (mTestigosAct.size() > 0)}
```

Invariante

```
{∀ tes : Testigo | tes ∈ mTestigosAct :
(∀ ng : NodoMSCGraph | ng ∈ tes.NodosRecorridos() : ng ∈ mMSCGraph.nodos())
 ∧ tes.NodosRecorridos() es un Camino de aceptación[6] en mMSCGraph}
//Todos los nodos recorridos de cada Testigo en mTestigosAct pertenecen a mMSCGraph y el orden en que
//aparecen determina un camino válido en mMSCGraph
```

Como se mencionó anteriormente, el tipo abstracto Testigo representa un CaTSE activo en el MSC-Graph analizado. Lleva consigo un BMSCMonitor, encargado del seguimiento del bMSC correspondiente al CaTSE en el MSC-Graph, y la secuencia de nodos recorridos correspondiente al CaTSE en el MSC-Graph.

La secuencia de nodos recorridos en un Testigo representa el conjunto de nodos que fueron visitados de la especificación MSC-Graph para reconocer la secuencia de etiquetas recibida. Es decir que los nodos junto con los ejes que los unen, representan el cubrimiento en el MSC-Graph obtenido de analizar la secuencia de etiquetas para este testigo.

Para las Pre y Post condiciones del TAD Testigo se utilizan los proyectores *virtuales* mNodosRecorridos que devuelve una Secuencia_NodoMSCGraph y representa la secuencia de nodos recorridos subyacente, mBMSCMonitor que representa el seguimiento del bMSC generado hasta el momento, y mMSCGraph que representa el MSCGraph analizado.

Cuando un proyector no es utilizado en las Pre y Post condiciones de una operación, debe interpretarse que no se ve modificado. Para utilizar los proyectores virtuales desde afuera del tipo, se utilizará el nombre del proyector sin la letra m, por ejemplo, para mNodosRecorridos se usará NodosRecorridos().

Las operaciones del TAD más importantes se describen a continuación:

[6] Como fue definido en la Definición 12.

tipo Testigo

operaciones más importantes

```
constructor(in mg : MSCGraph)
{True}
{mNodosRecorridos = {mg.getNodoInicial()} ∧ mBMSCMonitor = ∅ ∧ mMSCGraph = mg}

//Concatena una secuencia de bMSCs al bMSC subyacente.
concatCamino(in cam: Camino)
{mNodosRecorridos = nodosRecorridos0 ∧ mBMSCMonitor = bMSCMonitor0 ∧
 (∀ n : nodo | n ∈ cam : n ∈ mMSCGraph)}
{mNodosRecorridos = nodosRecorridos0 • cam ∧
 mBMSCMonitor = bMSCMonitor0.concatBMSC(cam)}
//La configuración actual se mantiene y el PO se amplía agregando el bMSC asociado a cada nodo recibido en
//el orden de la secuencia de nodos recibida (camino)

//Devuelve el último nodo recorrido
getNodoActual()                                  → ret_value : NodoMSCGraph
{True}
{ret_value = mNodosRecorridos.ult()}

//Busca los caminos hacia evento e de mensaje tal que l(e) = et y que sean válidos según el lenguaje denotado
//por el MSCGraph asociado al Testigo
buscarExtendido(in et: Etiqueta)                 → ret_value : Secuencia_Camino
{True}
{(∀ c : Camino | c ∈ ret_value : c es un CHE et en mMSCGraph)}
//Para todo camino resultante, cumple con ser un Camino hacia la etiqueta et según la definición de la sección
//4.4.1.
```

Invariante

$\{M_{mNodosRecorridos} = mBMSCMonitor.bMSC\}$

//La concatenación de todos los bMSCs de los nodos de *mNodosRecorridos* = bMSC de *mBMSCMonitor*

Al TAD BMSCMonitor se le agrega una nueva operación con respecto a su definición anterior, para poder concatenarle nuevos bMSCs al que el bMSC subyacente ya tenía.

tipo BMSCMonitor

operaciones agregadas

```
//Se concatena al bMSC subyacente, el bMSC asociado a cada nodo de la secuencia recibida,
//en el orden de la misma
concatBMSC(in cam: Camino)
{mBMSC = bMSC0}
{mBMSC = bMSC0 &b Mcam}
//Observación: en la concatenación no se ven afectadas la configuración ni la secuencia recorrida
//hasta el momento

//Verifica si existe algún evento pendiente de ser ejecutado en la instancia recibida. La instancia debe ser válida
ExistenEvPendEnInst?(in inst: Instancia)         → ret_value : Bool
{inst ∈ bMSC.I}
{ret_value = (∃ e : evento | mBMSC.g(e) = inst ∧ e ∈ mBMSC ∧ e ∉ Conf)}
```

El tipo abstracto MSCGraph tendrá una operación llamada "getInstanciaDeEtiqueta", tal que recibe como parámetro una etiqueta, y devuelve la instancia en el bMSC que corresponda, del evento que matchea con la etiqueta recibida. Esta operación será utilizada en el algoritmo `esPrefijo?`.

4.4.3 Algoritmo

La operación "esPrefijo?" del algoritmo, determina si una secuencia de etiquetas es un prefijo válido del lenguaje denotado por el MSC-Graph que especifica el comportamiento del sistema a monitorear.

La operación recorre la secuencia de etiquetas recibidas y por cada una de las etiquetas, recorre los testigos activos con el objetivo de extenderlos para que acepten la secuencia recorrida.

El proceso de los testigos activos con cada etiqueta de la secuencia se realiza en la operación procesarEtiqueta. Esta procesa la etiqueta ciclando por todos los testigos activos y para cada uno de ellos determina si se lo elimina o se lo utiliza como prefijo de nuevos testigos.

```
//Determina si una secuencia de etiquetas es prefijo válido del lenguaje denotado por el MSCGraph subyacente
esPrefijo?(in gr: MSCGraph, in s: Secuencia_Etiqueta)        → ret_value : Bool
{True}
{ret_value = (∃ C : Camino de aceptación | C es Camino de aceptación en gr ∧ s ∈ Lin_p(M_C)}
```

```
esPrefijo? (in gr: MSCGraph , in s: Secuencia_Etiqueta)      → ret_value : Bool
  var
    matchea          : Bool
    etiquetaActual   : Etiqueta
    testigos         : Testigos    //Representa los testigos activos durante la ejecucion del algoritmo
  begin

    matchea := true

    //Inicializa testigos con un Testigo (bMSC vacío) para comenzar a recorrer la secuencia de EtiquetaTS
    testigos := Testigos.constructor(gr)

    while (not s.esVacia?() and matchea)

        //Se obtiene la próxima Etiqueta, y se avanza el iterador
        etiquetaActual := s.prim()

        //Se procesan todos los testigos activos con la etiqueta actual
        procesarEtiqueta(testigos, etiquetaActual)

        //Si no existen testigos activos la secuencia no es prefijo del lenguaje
        matchea := testigos.existenTestigosActivos()

        //Se avanza a la próxima etiqueta
        s := s.fin()

    endwhile //De la secuencia de entrada

    //Si la secuencia es prefijo, debería existir algún testigo activo
    ret_value := matchea

  end
```

```
//Procesa una etiqueta ciclando por todos los testigos activos y para cada uno de ellos determina
//si se lo elimina o se lo utiliza como prefijo de nuevos testigos
procesarEtiqueta(in testigos: Testigos, in etiqActual: Etiqueta)
{testigos = testigos0}
{(∀ t : Testigo | t ∈ testigos.TestigosAct() :
 (∃ t' : Testigo | t' ∈ testigos0.TestigosAct() : t' es prefijo de t))}
```

```
procesarEtiqueta(in testigos: Testigos, in etiqActual: Etiqueta)
  var
    secCam          : Secuencia_CaminoTestigo
    testigoActual   : Testigo
    evMatch         : Evento
    i               : Integer
  begin

    //Se arranca la secuencia de Testigos a recorrer
    testigos.arrancarTestigos()

    //Se analiza la etiqueta actual para cada Testigo de Testigos
    while (testigos.hayProximo())
        //Se obtiene el próximo testigo y se avanza el iterador
        testigoActual := testigos.proximoTestigo()

        if (testigoActual.BMSCMonitor().existenEvPendEnInst?(
            etiqActual.getInstancia())) then
            //Existen eventos a ser ejecutados dentro del Testigo...
            //Se busca el evento que matchee y si existe, se avanza. Si no, se elimina el testigo

            //Se obtiene el evento actual que matchee con la etiqueta en caso que exista, sino retorna null
            evMatch := testigoActual.BMSMonitor().buscar(etiqActual)

            if (evMatch != null) then
                //Se avanza sobre el evento resultante en el testigo actual
                testigoActual.BMSMonitor().avanzar(evMatch)
            else
                //Se elimina el testigo actual ya que existen eventos pendientes en el bMSC subyacente,
                //pero no matchean con la etiqueta recibida. La ejecución por esta rama es inválida
                testigos.eliminarTestigo()
            endif

        else //No existen eventos a ser ejecutados dentro del Testigo.

            //Se buscan secuencia(Camino) asociados a la etiqueta actual con la configuración actual.
            //Para todo camino resultante, el primer nodo es un hijo de testigos.getTestigoActual().getNodoActual()
            secCam := testigoActual.buscarExtendido(etiqActual)

            if (not secCam.esVacia?()) then
                //Extiende el testigo actual con los nuevos caminos que surgen a partir del buscarExtendido.
                //Esto agrega los nuevos testigos en la Parte Derecha del iterador de testigos, de manera que
                //se avance localmente en cada uno de ellos a continuación
                testigos.extenderTestigo(secCam)

                //Se consume el evento que matchea en cada testigo nuevo resultante
                for (i:=0; i<secCam.size(); i++)
                    //Existen eventos a ser ejecutados dentro del Testigo...
                    //Se busca el evento que matchee y si existe, se avanza. Sino, se elimina el testigo

                    //Se obtiene el evento actual que matchee con la etiqueta en caso que exista, sino retorna null
                    evMatch := testigoActual.BMSMonitor().buscar(etiqActual)

                    if (evMatch != null) then
                       //Se avanza sobre el evento resultante en el testigo actual
                       testigoActual.BMSMonitor().avanzar(evMatch)
                    else
```

```
                    //Se elimina el testigo actual ya que existen eventos pendientes en el bMSC subyacente,
                    //pero no matchean con la etiqueta recibida. La ejecución por esta rama es inválida
                    testigos.eliminarTestigo()
                endif

                //Se avanza al próximo testigo
                testigos.avanzarTestigo()

                //Se obtiene el próximo testigo
                testigoActual := testigos.proximoTestigo()
            endfor

        else //secCam.size() == 0

            //Se elimina el testigo actual ya que no existe ningún camino hacia
            //la etiqueta basado en la configuración actual.
            testigos.eliminarTestigo()
        endif

    endif

    //Se avanza al próximo testigo
    testigos.avanzarTestigo()
endwhile
```

4.4.4 Prueba de corrección

El algoritmo para determinar si una secuencia de etiquetas es prefijo de un MSC-Graph consiste en utilizar múltiples testigos para reconocer diferentes posibles interpretaciones de la secuencia de etiquetas recibidas.

Para esto se intenta avanzar en cada testigo existente, extendiendo en caso que se pueda, a partir de la configuración actual que posee el mismo y los CHEs hacia la etiqueta actual de la secuencia.

Se analizan a continuación los dos ciclos que pueden observarse en el algoritmo, uno para recorrer la secuencia de etiquetas recibida (`esPrefijo?`) y otro para recorrer los testigos activos de la secuencia analizada (`procesarEtiqueta`).

Ciclo Externo: Recorriendo la secuencia de etiquetas:

1. **Condición de fin**
   El ciclo siempre termina porque se itera sobre la secuencia de etiquetas, y en cada paso se consume una etiqueta, decrementando el tamaño de la parte derecha de la secuencia.
   a. Es posible que el ciclo termine por no existir más etiquetas para consumir en la secuencia *s*.

   b. También es posible que se termine el ciclo en caso que no exista ningún testigo activo, ya que la secuencia no sería válida (*matchea* = False).

2. **Invariante**
   a. ($\forall$ *t* : Testigo | *t* $\in$ *Testigos*.TestigosAct() : PI(*s*) $\in$ Lin_c(M') | M': bMSC denotado por la configuración *t*.BMSCMonitor().Conf())
   La secuencia tratada hasta el momento es linearización completa del bMSC denotado por la configuración actual de cada uno de los CaTSEs activos.

   b. { *t*.NodosRecorridos() | *t* $\in$ *Testigos*.TestigosAct() } = { *c* : Camino de aceptación en G | *c* es un CaTSE PI(s) en G }
   Testigos.TestigosAct() es exactamente el conjunto de CaTSEs PI(*s*) en G.

El algoritmo termina cuando se sale del ciclo externo.

Ciclo Interno: Recorriendo todos los testigos para una etiqueta en particular (operación *procesarEtiqueta()*):

1. **Condición de fin**
   El ciclo termina porque se itera sobre la secuencia de testigos activos, y en cada paso se avanza con el iterador de testigos, decrementando el tamaño de la parte derecha.

   Notar que en caso que se encuentren nuevos testigos posibles y se agreguen a Testigos() durante una iteración, se avanza sobre cada uno de ellos, de manera que el tamaño de la parte derecha del iterador termina decrementándose en uno al llegar al fin del ciclo.

2. **Invariante**
   Sea $Testigos_0$ el conjunto de testigos de PI(s) antes de entrar en el ciclo interno.
   $\exists$ PI_0, PD_0 tal que $Testigos_0$.TestigosAct() = PI_0 & PD_0.
   Testigos.TestigosAct() = PI & PD
   PD = PD_0.
   PI = { *t* : Testigo | *t*.NodosRecorridos() es un CaTSE (PI(s) • s.prim()) en G $\wedge$
   ($\exists$ *t'* : *t'* CaTSE PI_0 en G | *t'* es prefijo de t) }

   Comentarios:
   Por la forma de construcción del algoritmo, todos los testigos que pertenecen a PI(*Testigos*.TestigosAct()) tienen como prefijo algún testigo de PI_0.
   Al final del ciclo, se procesó todo $Testigos_0$.TestigosAct(), de manera que PD_0 = {}. Luego, por Observación 1, se sabe que todo testigo de *Testigos*.TestigosAct() tiene como prefijo algún testigo de $Testigos_0$.TestigosAct().

El ciclo externo funciona porque se parte de los CaTSEs de la PI(s) y la construcción del algoritmo se basa exclusivamente en tratar de conseguir nuevos CaTSEs partiendo de ellos, lo que garantiza que al finalizar el ciclo se obtengan los CaTSE (PI(s) • s.prim()) (por Observación 1).

Observación 1: Dado MSC-Graph G, ($\forall$ C CaTSE (S • *a*) en G ($\exists$ C' CaTSE S en G que es prefijo de C))

Para validar que la observación es cierta, se analizan los dos posibles casos para C CaTSE (S • *a*):

1. El evento que matchea con *a* es un evento minimal en el último bMSC (correspondiente al último nodo) de C.
   Para obtener un C' CaTSE S, se deben eliminar todos los nodos concatenados para formar el CaTSE (S • *a*), que se obtuvieron en la búsqueda de los eventos que matchean con la etiqueta *a* (sería un rollback del CHE *a*). De esta manera se consigue C' CaTSE S sin existencia de un sufijo no utilizado y prefijo de C.

2. El evento que matchea con *a* NO es un evento minimal en el último bMSC (correspondiente al último nodo) de C.
   Aquí pueden suceder dos cosas:
   - Existe más de un evento en la instancia correspondiente al evento que matchea con *a* en el último bMSC de C.
   - El evento que matchea con *a* se encuentra en algún bMSC intermedio (correspondiente a algún nodo intermedio) de C.

   En este caso, para obtener un C' CaTSE S no es necesario modificar C, es decir que eliminar de la secuencia a la etiqueta *a* no impacta en C y por lo tanto C es válido como CaTSE S (no existe un sufijo no utilizado porque los bMSCs usados son los mismos que cuando se necesite matchear con la etiqueta *a*).

4.5 Monitoreo de MSC-Graphs

Se presenta en esta sección una solución al problema de analizar posibles ejecuciones de un sistema, para validar su correctitud de acuerdo al comportamiento especificado a nivel MSC-Graph con restricciones de tiempo y sin CIDs.

Se muestra un ejemplo de MSC-Graph con restricciones de tiempo que se utilizará para mejorar la comprensión de los conceptos introducidos:

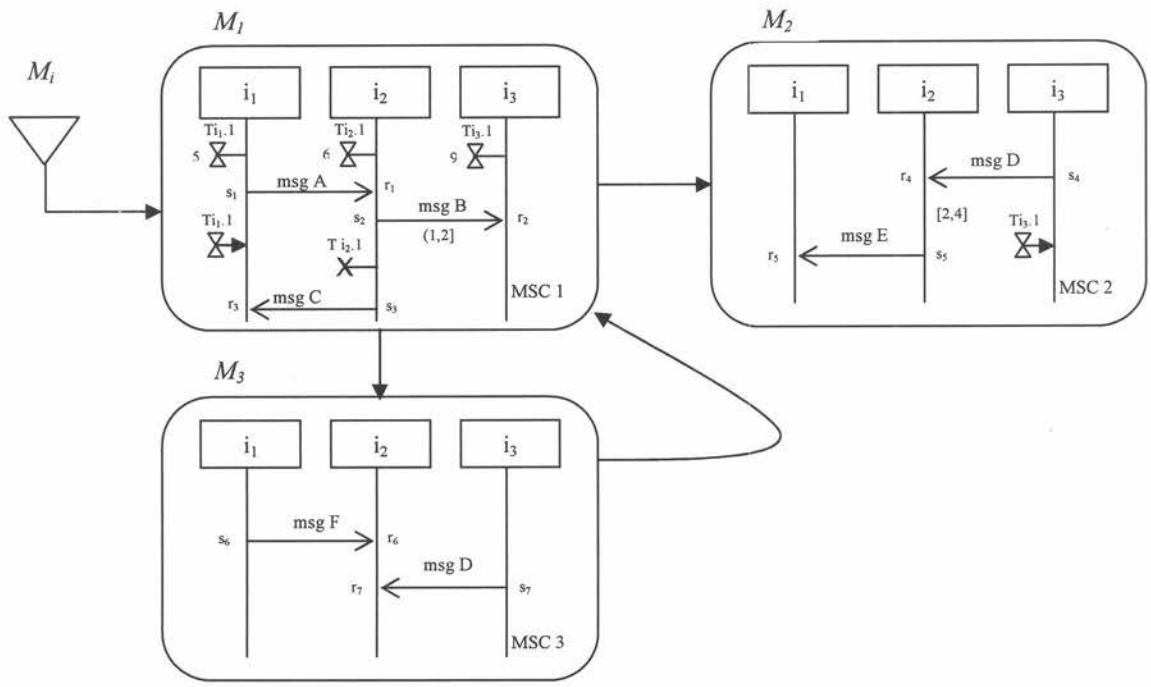


Figura 15

4.5.1 Definiciones

En la sección 3.3.2 "Semántica de bMSC" se han definido las condiciones de tiempo que deben tener los eventos para que sean válidos según el lenguaje denotado por un bMSC.

En la "Definición 9: Concatenación de dos bMSCs" se ha definido el resultado de la concatenación de dos bMSCs.

Se recuerda la "Definición 16: Camino testigo de una secuencia de etiquetas σ: CaTSE" y la "Definición 17: Camino hacia una etiqueta et: CHE" para ser utilizadas con bMSCs con restricciones de tiempo, aplicando en las definiciones los bMSCs con restricciones de tiempo asociados a los nodos del MSC-Graph en lugar de bMSCs.

4.5.2 Analizando MSC-Graphs

Los MSC-Graphs con restricciones de tiempo, difieren de su par sin temporizar, en que poseen restricciones dadas por Timers y/o Delay Intervals que deben ser verificadas.

El monitoreo de estos MSC-Graphs consiste en verificar las mismas condiciones sobre los eventos que se realizan al monitorear un MSC-Graph sin restricciones de tiempo, incorporando ahora la verificación sobre ellas, tal que los eventos cumplan con las mismas. Para poder verificarlas, se deben evaluar los Timestamps de cada etiqueta y llevar un estricto control de las restricciones para cada instancia y mensaje.

TADs utilizados

A continuación se detallan los formalismos y el algoritmo asociado para realizar el monitoreo de sistemas especificados por MSC-Graphs con restricciones de tiempo.

A diferencia del algoritmo de monitoreo de MSC-Graphs sin restricciones de tiempo, se modifican algunas operaciones que deben encapsular las validaciones sobre las restricciones de tiempo, entre ellas las que aplican los Timers y Delay Intervals.

Al TAD BMSCMonitor se le modifican las operaciones `buscar` y `avanzar`. La primera, al realizar la búsqueda del evento de mensaje que matchee con la etiqueta recibida de la secuencia, debe validar si el Timestamp asociado a la etiqueta cumple con las restricciones de tiempo de Timers, Delay Intervals y crecimiento en el tiempo, definidos en la configuración del BMSCMonitor sobre las instancias y/o los mensajes en los cuales ejerce el evento.

La operación `avanzar` en lugar de recibir un evento, recibe ahora un evento temporizado para extender la configuración del BMSCMonitor. El evento es el que matchea con la etiqueta recibida en la operación `buscar` y el Timestamp es el asociado a la etiqueta. Debe agregar la etiqueta a la secuencia recorrida y agregar a la configuración el evento que matchea con ella, junto con todos los eventos de Timers que se encuentren entre el evento a agregar y el último ocurrido sobre la misma instancia.

La operación `buscarExtendido` no posee modificaciones ya que en ella no se realizan verificaciones de tiempo. Por la forma del algoritmo `esPrefijo?`, las verificaciones de tiempo siempre se realizan localmente en la operación `buscar` del TAD BMSCMonitor.

Como se había comentado con anterioridad, los eventos de Timers no son visibles en la ejecución de un sistema. Por lo tanto, según $Comp_t$, la verificación de las restricciones de los Timers implica analizar si se pueden agregar eventos de Timers tal que sus restricciones de tiempo son satisfechas por los eventos temporizados.

Para simplificar el análisis, en la mayoría de los ejemplos se puede asumir que los seteos de Timers no están libres, sino que se pueden asociar al evento inmediatamente anterior. En particular en los casos de estudio analizados, esto no implica problema alguno y simplifica el procedimiento. En la sección "8 Trabajos a futuro", se comenta la idea para trabajar con la versión completa, que no sería muy distinto a analizar un problema de programación lineal sencillo.

Se decidió definir $Comp_t'$, una versión más fuerte que $Comp_t$, que se utilizará en los algoritmos y que permitirá simplificar el proceso de verificación de las restricciones de tiempo de Timers.

Formalmente:

Definición 20: $Comp_t'(\sigma, C, M_t)$ Comportamiento' válido para una configuración de un bMSC

$Comp_t'(\sigma, C, M_t) \Rightarrow Comp_t(\sigma, C, M_t)$ (Ver Definición 6) con un agregado a las condiciones ya definidas sobre la secuencia de eventos temporizada proyectada a cada una de las instancias.

$\forall$ i : instancia, α_i : seq(EventoTS) proyectada a la instancia i cumple:

$\forall\ \alpha_i[j] = (e_j, ts_j) \wedge tipoEv(e_j) = \theta \wedge OpTimer(e_j) = Set$

si $0 < j < \#\alpha_i \quad \Rightarrow ts_j = ts_{j-1}$

si $j = 0 \quad \Rightarrow ts_j = 0$

Comentario:

- Se setean los Timestamps de los eventos Set de Timer (invisibles en el log de ejecución) al evento inmediatamente anterior. Esto quiere decir que a todo evento de Timer se le asocia el Timestamp correspondiente al evento anterior.
- En caso de que no exista un evento anterior, los Timestamps de los eventos Set de Timer serán cero.
- Tener en cuenta que deben cumplirse todas las restricciones sobre σ definidas en $Comp_t$.

Entonces el algoritmo `buscar` no chequeará exactamente las condiciones que se describen en la Definición 6, sino que validará algo más fuerte descripto por la Definición 20.

```
tipo BMSCMonitor
        operaciones modificadas

        //Busca los Eventos e asociados a mensajes tales que l(e) = h, que su Timestamp sea válido
        //según las restricciones de tiempo (Timers, Delay Intervals, crecimiento) definidas y que
        //pueden extender la configuración del bMSC. Se sabe que existirá ninguno o un evento.
        buscar(in h: EtiquetaTS)                    → ret_value : Secuencia_EventoTS
        {mConfig = conf0 ∧ mSeqRec = seqRec0}
        {∀ ev ∈ ret_value : ev.getEvento() ∈ {S ∪ R} ∧ mBMSC.pertenece?(ev.getEvento()) ∧
         mBMSC.getLabel(ev.getEvento()) = h.getLabel() ∧ ev.getTStamp() = h.getTStamp() ∧
         conf = conf0 ∪ {e: Evento | tipoEv(e)= θ ∧ e < ev.getEvento()} ∪ {ev.getEvento()}
         ∧ mBMSC.esConfiguracion?(conf) ∧ Compt'(seqRec0 • h, conf, mBMSC)}
```

```
    ∧ mBMSC.esConfiguracion?(conf) ∧ Comp_t'(seqRec_0 • h, conf, mBMSC)}

    //Extiende la configuración del bMSC con el evento recibido y todo evento de timer anterior al mismo
    avanzar(in ev: EventoTS)
    {mConfig = conf_0 ∧ mSeqRec = seqRec_0 ∧
     Comp_t'(mSeqRec • mBMSC.getLabel(ev.getEvento()), mConfig, mBMSC) ∧
     mBMSC.esConfiguracion?(mConfig ∪ {e : Evento | tipoEv(e) = θ ∧
     e < ev.getEvento()} ∪ {ev.getEvento()})}
    {mConfig = conf_0 ∪ {e : Evento | tipoEv(e) = θ ∧ e < ev.getEvento()} ∪
     {ev.getEvento()}) ∧ mSeqRec = seqRec_0 • mBMSC.getLabel(ev.getEvento())}
```

Se utiliza el TAD EtiquetaTS que representa una etiqueta con su Timestamp asociado. Posee dos operaciones: **getLabel()** para obtener la etiqueta y **getTStamp()** para obtener su Timestamp correspondiente. Lo mismo ocurre con el TAD EventoTS que posee como operaciones **getEvento(), getTStamp(), setTStamp(Timestamp)** y obtienen en forma correspondiente el evento, el Timestamp y el seteo de un Timestamp.

Notar que el TAD BMSCMonitor asegura siempre por construcción, que la secuencia recorrida hasta el momento satisface $Comp_t'$.

4.5.3 Algoritmo

El nuevo algoritmo consta de la operación "`esPrefijo?`", que determina si una secuencia de etiquetas temporizadas es un prefijo válido del lenguaje denotado por el MSC-Graph con restricciones de tiempo que especifica el comportamiento del sistema a monitorear.

El algoritmo es igual al descripto en la sección anterior porque no requiere modificaciones con respecto a su estructura.

Se produce un cambio de interfaz tanto en el algoritmo `esPrefijo?`, como en la operación `procesarEtiqueta`. El algoritmo recibe ahora una secuencia de etiquetas temporizada en lugar de una secuencia de etiquetas, y la operación una etiqueta temporizada en lugar de una etiqueta. Lo mismo ocurre internamente porque utilizarán ahora eventos temporizados en lugar de eventos.

Se debe notar que en las post condiciones de estas operaciones, ahora se aplica la linearización parcial definida en la sección "3.3.2 Semántica de bMSC", que se refiere a la definición de bMSCs con restricciones de tiempo.

El único cambio realizado es la operación `buscar` del TAD BMSCMonitor, que también cambia su interfaz al recibir una etiqueta temporizada. Esta operación debe verificar que se cumplan las restricciones de tiempo definidas en Definición 20 y especificadas en el MSC-Graph del sistema monitoreado. Puede consultarse en la sección "4.5.5 Algoritmos auxiliares".

```
//Determina si una secuencia de etiquetas temporizadas es prefijo válido del lenguaje denotado por el
//MSCGraph subyacente
esPrefijo?(in gr: MSCGraph, in s: Secuencia_EtiquetaTS)          → ret_value : Bool
{True}
{ret_value = (∃ C : Camino de aceptación | C es Camino de aceptación en gr ∧ s ∈ Lin_p(M_C)}
```

4.5.4 Prueba de corrección

El algoritmo para determinar si una secuencia de etiquetas temporizadas es prefijo de un MSC-Graph con restricciones de tiempo consiste en utilizar múltiples testigos para reconocer diferentes posibles interpretaciones de la secuencia de etiquetas temporizadas recibidas.

Para esto se intenta avanzar en cada testigo existente, extendiendo los mismos en caso que se pueda, a partir de la etiqueta temporizadas de la secuencia de entrada y la configuración actual que posee el mismo.

Ambos ciclos (interno y externo) del algoritmo poseen las mismas características analizadas que en el algoritmo anterior, por lo tanto son los mismos que se analizaron sin la verificación de restricciones de tiempo y sus condiciones de fin son las mismas. La única diferencia radica en que se deberá asegurar en los invariantes de los ciclos, las condiciones que deben cumplir los Timestamps de las etiquetas de la secuencia, sobre las restricciones de tiempo del MSC-Graph (es decir cumplir $Comp_t'$):

Ciclo Externo
($\forall$ *t* : Testigo | *t* $\in$ *Testigos*.TestigosAct() : $Comp_t$'(PI(s), *t*.BMSCMonitor().Conf(), *t*.BMSCMonitor().BMSC()))
La parte tratada hasta el momento de la secuencia de etiquetas temporizadas, representa un comportamiento válido (según la Definición 20) para la configuración del bMSC subyacente a cada uno de los testigos activos.

Ciclo Interno
($\forall$ *t* : Testigo | *t* $\in$ PI(*Testigos*.TestigosAct()) : $Comp_t$'(PI(s) • s.prim(), *t*.BMSCMonitor().Conf(), *t*.BMSCMonitor().BMSC()))
La parte tratada hasta el momento de la secuencia de etiquetas temporizadas (PI(s) • s.prim()), representa un comportamiento válido (según la Definición 20) para la configuración del bMSC subyacente a cada uno de los testigos recorridos (PI de testigos activos).

Todas las restricciones de tiempo que deben cumplir los Timestamps de las etiquetas serán validadas en la operación `buscar`, que se describe en la próxima sección.

4.5.5 Algoritmos auxiliares

Definiciones

Las siguientes definiciones son utilizadas a lo largo de la sección, especialmente en el análisis de las restricciones de tiempo y en los algoritmos auxiliares que se describen para validar las mismas.

Definición 21 : Timer presente

Dado *M* : bMSC; *t* : Timer se dice presente en una configuración C de *M* sii
($\exists$ *f*: Evento | *C*.pertenece?(*f*) $\wedge$ tipoEv(*f*) = θ $\wedge$ GetTimer(*f*) = t)

Comentario:
- Dada una configuración C de un bMSC, un Timer t se dice presente, si existe algún evento de t en C.

Definición 22 : Timer abierto

Dado *M* : bMSC; *t* : Timer presente se dice abierto en una configuración C de *M* sii
($\exists$ *f*: evento | *C*.pertenece?(*f*) $\wedge$ tipoEv(*f*) = θ $\wedge$ GetTimer(*f*) = *t* $\wedge$ OpTimer(*f*) = Set $\wedge$
$\neg$ ($\exists$ *h*: Evento | *C*.pertenece?(*h*) $\wedge$ tipoEv(*h*) = θ $\wedge$ GetTimer(*f*) = GetTimer(*h*) $\wedge$
(OpTimer(*h*) = Timeout $\vee$ OpTimer(*h*) = Reset) $\wedge$ *f* < *h*))

Comentario:
- Dada una configuración C de un bMSC, un Timer se dice activo en la configuración, si existe algún evento de Set que ya sucedió, y aún no ha sucedido su Reset o Timeout (en particular será el último Set, ya que según la Definición 6, para cada Set anterior al último, como existe al menos un Set después, no cumple con la condición de mantenerlo Activo).

Definición 23 : Timer expiró por Timeout

Dado *M* : bMSC; *t* : Timer presente se dice que expiró por Timeout en una configuración C de *M* sii
($\exists$ *f*, *h*: evento | *C*.pertenece?(*f*) $\wedge$ *C*.pertenece?(*h*) $\wedge$ tipoEv(*f*) = tipoEv(*h*) = θ $\wedge$ GetTimer(*f*) = GetTimer(*h*) = *t* $\wedge$
OpTimer(*f*) = Set $\wedge$ OpTimer(*h*) = Timeout $\wedge$ *f* < *h* $\wedge$ $\neg$ ($\exists$ *j*: Evento | *C*.pertenece?(*j*) $\wedge$ tipoEv(*j*) = θ $\wedge$
GetTimer(*f*) = GetTimer(*j*) $\wedge$ OpTimer(*j*) = Set $\wedge$ *h* < *j*))

Comentario:
- Dada una configuración C de un bMSC, un Timer se dice que expiró por Timeout, si existe un par de eventos de Set y Timeout, tal que el Timeout ocurre luego del Set, y además, no existe ningún evento posterior al del Timeout que sea un Set.

TADs utilizados

Las estructuras utilizadas para elaborar los algoritmos son un bMSC y una configuración. Para una mejor comprensión de los algoritmos auxiliares, se detalla a continuación la configuración que se representará con el TAD *BMSCConfig*. Su descripción no fue necesaria con anterioridad porque en las secciones previas no se requería un importante uso del mismo y se la podía asociar con el concepto matemático de configuración.

El TAD *BMSCConfig* extiende el concepto de configuración original porque no sólo permitirá mantener actualizada la configuración de un bMSC, sino que brindará información adicional para facilitar las validaciones de las restricciones de tiempo establecidas por los Timers.

Para ello posee el bMSC sobre el cual lleva la configuración y los estados actuales de cada una de las instancias del mismo. Se describe el estado de una instancia como:

1. El último evento ejecutado en la instancia.
2. El último evento de $S \cup R$ ejecutado en la instancia.
3. El próximo posible evento de $S \cup R$ a ejecutar en la instancia.
4. Los eventos de Timers activos en la instancia.

Un Timer activo puede ser un Timer abierto según la Definición 22, o un Timer que expiró por Timeout según la Definición 23, descriptas con anterioridad.

Recordar que todos los eventos tienen un Timestamp asociado, en particular los eventos de Timers.

El conjunto formado por el último evento de $S \cup R$ ejecutado en cada una de las instancias, es el concepto de configuración original. Los eventos anteriores en cada instancia (sin importar si es un evento de $S \cup R$), los próximos eventos posibles y los Timers activos, constituyen la información adicional.

El invariante de representación de *BMSCConfig* determina que posee estados válidos de las instancias, sobre los eventos del bMSC asociado.

Para acceder a la información del estado de una instancia, cuenta con los siguientes proyectores:

1. La operación `getUltEvMarcadoEnInst`, dada una instancia, devuelve en caso que exista, el último evento que pertenece a la instancia en la configuración.
2. La operación `getUltEvMsgMarcadoEnInst`, dada una instancia, devuelve en caso que exista, el último evento de mensaje que pertenece a la instancia en la configuración.
3. La operación `getEvMsgProxEnInst` permite obtener dada una instancia, el próximo evento de $S \cup R$ a ejecutarse en ella y que aún no pertenece a la configuración. En el caso que para una instancia no exista un próximo evento a ser ejecutado, la operación no devuelve ningún evento.
4. La operación `getEventosDeTimerActivos`, permite obtener dada una instancia, una secuencia de eventos $\in \theta$, que representan aquellos Timers activos en la configuración (según Definición 22 y Definición 23).

Además provee como operaciones básicas el `buscar` y `avanzar`.
La operación `avanzar` se encarga de actualizar el estado de la instancia correspondiente al evento por el cual se desea avanzar. Realiza los siguientes pasos:

1. Actualiza la configuración, agregando el evento recibido como el último evento ejecutado en la instancia.
2. Agrega el evento recibido como el último evento de $S \cup R$ ejecutado en la instancia, si corresponde.
3. Setea el próximo evento de $S \cup R$ posible a ser ejecutado en la instancia correspondiente.
4. Actualiza los Timers activos, agregando los eventos de Timer que se encuentren definidos entre el evento recibido y el próximo evento a ser ejecutado en la instancia correspondiente.

Se describe a continuación la estructura de representación del tipo, utilizada para la solución y necesaria para comprender los algoritmos que se presentarán.

BMSCConfig == tupla < *mEstadoInsts*: Map_EstadoInstancia , *mBMSC* : BMSC >

El TAD *BMSCConfig* posee un primer componente llamado *mEstadoInsts* que representa para cada instancia el estado en el que se encuentra (mapping): último evento ejecutado, próximo evento y Timers activos. Además posee *mBMSC* que referencia al BMSC sobre cual se lleva la configuración. Será el BMSC que posee *BMSCMonitor*.

El invariante de representación de *BMSCConfig* determina que *mEstadoInsts* posee EstadoInstancia(s) válidos sobre Eventos de *mBMSC*.

tipo BMSCMonitor

operación modificada

//Busca los eventos que pueden extender la configuración del bMSC, concuerdan con la etiqueta temporizada //ets y cumplen las restricciones de tiempo establecidas

```
buscar(in ets: EtiquetaTS)                    → ret_value : Secuencia_EventoTS
```

//Las Pre y Post condiciones fueron descriptas con anterioridad en la presentación de operaciones modificadas //de la sección 4.5.2 Analizando MSC-Graphs.

```
buscar(in ets: EtiquetaTS)                    → ret_value : Secuencia_EventoTS
    var
        validoTemp       : Bool
        inst             : Instancia
        ultEv            : EventoTS
        proxEvMsg        : EventoTS
        evTimer          : EventoTS
        evTimerSet       : EventoTS
        hmTimersEnInst   : Map_Timers
        config           : BMSCConfig
        seqEvTimers      : Secuencia_EventoTS
    begin

        ret_value := Secuencia_EventoTS.constructor()

        //Se obtiene la instancia de la etiqueta
        inst := ets.getInstancia()

        //Se obtiene el próximo EventoDeMensaje en la instancia de la etiqueta
        proxEvMsg := mConfig.getEvMsgProxEnInst(inst)

        //Se verifica si el próximo evento de la instancia es de mensaje y la
        //etiqueta que tiene asociada es igual a la recibida como parámetro
        if (proxEvMsg != null) then

            //Se continua si matchea, para verificar si se cumplen las restricciones de tiempo
            if (mBMSC.getLabel(proxEvMsg) == ets.getLabel()) then

                //Se obtiene el último evento ejecutado en la instancia
                ultEv := mConfig.getUltEvMarcadoEnInst(inst)

                //Se obtiene la secuencia de eventos de Timer entre ultEv y proxEvMsg
                seqEvTimers := mBMSC.getEvTimers(ultEv, proxEvMsg)

                //Se obtiene una nueva configuracion para verificar que al agregarse los eventos de Timers entre
                //el último evento en la misma instancia y proxEvMsg, siga siendo válida temporalmente
                config := mConfig

                validoTemp := true

                //Se recorre la secuencia de eventos de Timers
                while (not seqEvTimers.esVacia?() ∧ validoTemp)

                    evTimer := seqEvTimers.prim()

                    //Se obtiene el último evento ejecutado en la instancia
                    ultEv := config.getUltEvMarcadoEnInst(inst)

                    //Se setea el Timestamp del evento de Timer según Comp_t', es decir asociándole el
                    //Timestamp del evento anterior ocurrido en la misma instancia
                    switch (evTimer.getTipo())
                            case Set:
                                    evTimer.setTStamp(ultEv.getTStamp())
                            case Reset:
                                    evTimer.setTStamp(ultEv.getTStamp())
```

```
                    case Timeout:
                        //Se obtiene los EvTimers activos para obtener el del Set correspondiente
                        hmTimersEnInst := config.getEventosDeTimerActivos(inst)
                        evTimerSet := hmTimersEnInst.get(evTimer.getTimer())
                        evTimer.setTStamp(evTimerSet.getTStamp() +
                                          evTimerSet.Dur())
                endswitch

                //Se valida temporalmente el evento contra la nueva configuración
                //Sólo es necesaria la verificación que satisface los Timers activos
                validoTemp := satisfaceTimers?(config, evTimer))

                //Se agrega el evento de Timer a la nueva configuración
                config := mConfig ∪ {evTimer}

                seqEvTimers := seqEvTimers.fin()
            endwhile

            if (validoTemp) then
                //Se setea el Timestamp al Evento
                proxEvMsg.setTStamp(ets.getTStamp())

                //Se valida temporalmente el evento contra la nueva configuración
                if (esValidoTemp?(config, proxEvMsg)) then
                    //Si todo salio bien, se agrega a la respuesta
                    ret_value.agregar(proxEvMsg)
                endif
            else
                Error: "No existe una secuencia de eventos Timestamped válida para el
                        bMSC subyacente"
            endif

        else
            Error: "Etiqueta no encontrada en bMSC subyacente"
        endif

    else
        Error: "No existen eventos posteriores en la instancia de la etiqueta"
    endif

    //Finalmente, ret_value tendrá algún EventoTS en caso que matchee y ninguno en caso contrario
end
```

La operación `buscar` debe verificar que se cumplan todas las restricciones de tiempo descriptas en $Comp_t$'. Para ello, utiliza la operación `esValidoTemp?`, encargada de validar que se cumplan las restricciones de tiempo de los Timers, de los Delay Intervals y el crecimiento en el tiempo. La operación `esValidoTemp?`, a su vez, utiliza las operaciones `satisfaceTimers?` y `satisfaceDIs?`, y realiza las verificaciones correspondientes en forma incremental, es decir que por cada nuevo evento a validar, no repite las verificaciones realizadas sobre los eventos previos.

Notar que se aplica al bMSC la operación `getEvTimers()`, que dados dos eventos, devuelve una secuencia formada por los eventos de Timers que se encuentren entre ellos y ordenada según la relación < del bMSC.

Análisis informal de restricciones de tiempo

A continuación se analiza individualmente la forma de proceder con cada tipo de restricción de tiempo y que las operaciones auxiliares deberán resolver:

Delay Intervals

Las restricciones sobre los DIs son relativamente simples de verificar, ya que siempre son entre un par de eventos consecutivos, ya sea a nivel instancia o a nivel mensaje. Por lo tanto, la restricción de tiempo debe verificarse al recibir el segundo evento. La restricción siempre se encuentra dentro de un mismo bMSC, y no existen DIs que sean Inter-bMSCs.

1. **Asociados a un eje o mensaje**

   Se analiza el mensaje msg B denotado por los eventos s_2-r_2 para el bMSC MSC1 del nodo M1 en la Figura 15. El mismo tiene un DI (1, 2].

   Para poder evaluar la restricción de este DI, es necesario registrar el Timestamp de s_2, y luego el de r_2.
   En el momento de la ocurrencia de r_2, se procede a verificar si $1 < t(r_2) - t(s_2) \leq 2$.
   Es decir que se valida $\Pi_1(1,2] < t(r_2) - t(s_2) \leq \Pi_2(1,2]$.

   En caso que esto no suceda, entonces el tiempo en el que ocurrió R(B) fue mayor o menor que el que permite la restricción de tiempo denotada por el DI. Si es así, la secuencia de entrada no conforma un comportamiento válido para el sistema especificado por la Figura 15.

2. **Asociados a la velocidad de procesamiento**

   Se analiza el intervalo entre r_4 y s_5 para el bMSC MSC2 del nodo M2 en la Figura 15. El mismo tiene un DI [2, 4].

   Para poder evaluar la restricción de este DI, es necesario registrar el Timestamp de r_4, y luego el de s_5.
   En el momento de la ocurrencia de s_5, se procede a verificar si $2 \leq t(s_5) - t(r_4) \leq 4$.
   Es decir que se valida $\Pi_1[2, 4] \leq t(s_5) - t(r_4) \leq \Pi_2[2, 4]$.

   En caso que esto no suceda, entonces el tiempo en el que ocurrió S(E) fue mayor o menor que el que permite la restricción de tiempo denotada por el DI. Si es así, la secuencia de entrada no conforma un comportamiento válido para el sistema especificado por la Figura 15.

Timers

En la Figura 16 se analizan las distintas combinaciones esquemáticas de Timers con mensajes:

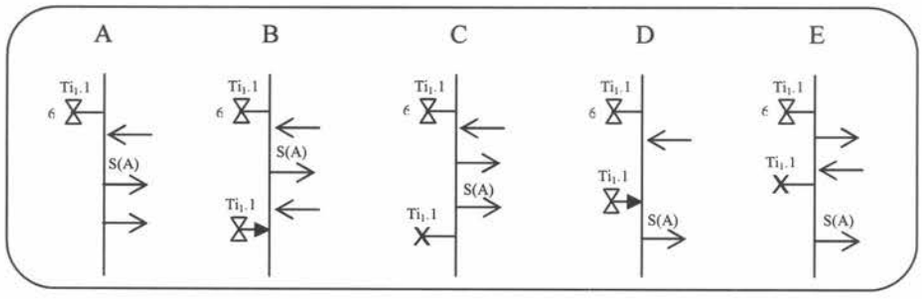


Figura 16

A, B y C: Se debe validar que $t(Ti_1.1) < t(S(A)) < t(Ti_1.1) + Dur(Ti_1.1)$.

D: Se debe validar que $t(Ti_1.1) + Dur(Ti_1.1) < t(S(A))$.

E: No debe validarse nada.

1. **Timer que expiró por Timeout** (Definición 23)

a. Eventos dentro del intervalo de Timer
Se analiza el Timer $Ti_1.1$ en la Figura 15: se setea $Ti_1.1$ antes de la ocurrencia del evento s_1 y con una duración 5 de unidades de tiempo.
Si se asocia el Timestamp del Timer al evento inmediatamente anterior, $Ti_1.1$ ocurre en Timestamp 0, entonces $0 \leq t(s_1) \leq 5$.

Es decir que se valida $t(Ti_1.1) \leq t(s1) \leq t(Ti_1.1) + Dur(Ti_1.1)$.

En caso que esto no suceda, entonces el tiempo en el que ocurrió S(A) fue mayor o menor que el que permite la restricción de tiempo denotada por el Timer $Ti_1.1$. Si es así, la secuencia de entrada no conforma un comportamiento válido para el sistema especificado por la Figura 15.

b. Eventos luego del intervalo de Timer
Además, todo evento que ocurra luego del Timeout del Timer, debe tener un Timestamp mayor que el Timestamp del evento de Timeout.
Se analiza el mismo Timer $Ti_1.1$ en la Figura 15, que se setea antes de s_1 y con una duración 5.
Si $Ti_1.1$ ocurre en el Timestamp 0 y el Timeout está luego de s_1 pero antes del Timestamp de r_3, entonces $5 \leq t(r_3)$.

Es decir que se valida $t(Ti_1.1) + Dur(Ti_1.1) \leq t(r_3)$.

En caso que esto no suceda, entonces el tiempo en el que ocurrió R(C) fue menor que el que permite la restricción de tiempo denotada por el Timer $Ti_1.1$. Si es así, la secuencia de entrada no conforma un comportamiento válido para el sistema especificado por la Figura 15.

2. **Timers cerrados con intervalo Set – Reset**

a. Eventos dentro del intervalo de Timer
Se debe realizar la misma validación descripta en el punto a de **Timer que expiró por Timeout.**

3. **Timer abierto** (Definición 22)

Cuando se encuentran Timers abiertos, se trata de cualquiera de los dos casos anteriores. Esto ocurre por la forma de operar de los algoritmos desarrollados, que concatenan los bMSCs del MSC-Graph. Al concatenar dos bMSCs M_a y M_b, donde M_a posee Set de un Timer abierto, y M_b un Reset/Timeout del mismo, se obtiene un nuevo bMSC con ambos eventos de Timer y se convierte en uno de los dos casos mencionados con anterioridad.

Como ejemplo, se analiza el Timer $Ti_3.1$ en la Figura 15: se setea $Ti_3.1$ antes de la ocurrencia del evento r_2 en el nodo M_1, suponiendo una duración 9 de unidades de tiempo. Sucede un Timeout luego de la ocurrencia del evento s_4 en el nodo M_2. Esto es un caso particular de Set-Timeout.
Se supone una ejecución tal que el camino actual consiste de $M_1 - M_3 - M_1 - M_2$.
Como $Ti_3.1$ ocurre dos veces y debido a la Definición 6Definición 6, se toma al segundo Set del Timer como un Reset+Set, tal que debido a que el Timestamp de Timer se asocia al evento anterior, $Ti_3.1$ en su segunda ocurrencia tiene el Timestamp de s_7, entonces: $t(Ti_3.1) = t(s_7)$.

$$0 < \{t(M1_1.r2), t(s7)\} < 9$$
$$t(s7) < \{t(M1_2.r2), t(s4)\} < t(s7) + 9$$

Es decir, que las restricciones de tiempo verificadas para el Timer $Ti_3.1$ en el camino M1 – M3 – M1 – M2, son:

$$t(\text{primer Set de } Ti_3.1) < \{t(M1_1.r2), t(s7)\} < t(\text{primer Set de } Ti_3.1) + Dur(Ti_3.1)$$
$$t(\text{segundo Set de } Ti_3.1) < \{t(M1_2.r2), t(s4)\} < t(\text{segundo Set de } Ti_3.1) + Dur(Ti_3.1)$$

La operación `esValidoTemp?` verifica en forma incremental $Comp_t'$, validando todas sus restricciones de tiempo definidas. Es decir, dada una secuencia de etiquetas que es un comportamiento válido según $Comp_t'$, verifica que al agregar una nueva etiqueta a la secuencia, la misma siga siendo un comportamiento válido para la configuración recibida.

tipo BMSCMonitor

operación agregada

//Determina si un Evento temporizado representa un comportamiento válido para la configuración recibida

```
esValidoTemp?(in config: BMSCConfig, in evMsg: EventoTS)    → ret_value : Bool
{mBMSC.esConfiguracion?(config) ∧ mBMSC.pertenece?(evMsg.getEvento())}
{ret_value = Comp_t'(mSeqRec • mBMSC.getLabel(evMsg), config, mBMSC)}
```

```
esValidoTemp?(in config: BMSCConfig, in evMsg: EventoTS)          → ret_value : Bool
    var
        evMsgAnt : EventoTS
        esValido : Bool
    begin

        esValido := true

        //Se valida que el TS sea mayor o igual que el del evento anterior (si existe) en la misma instancia
        evMsgAnt := config.getUltEvMarcadoEnInst(evMsg.getInstancia())
        if (evMsgAnt != null) then
            esValido := (evMsg.getTStamp() >= evMsgAnt.getTStamp()
        endif


        //Se valida que el TS sea mayor o igual que el del evento hermano correspondiente a la misma Etiqueta
        //Esta validación sólo debe realizarse si el evento actual tiene orientación Receive (R)
        if (esValido && evMsg.getTipo() == R) then
            evMsgAnt  := evMsg.getEvMsgHer()
            esValido := (evMsgAnt.getTStamp() ≤ evMsg.getTStamp())
        endif

        //Se valida que si hay DIs, se cumplan
        if (esValido) then
            esValido := satisfaceDIs(config, evMsg)
        endif

        //Se valida que si hay Timers abiertos, su TS sea menor que TS de Set+Dur
        //También que si hay un Timer que haya sido recién cerrado, el TS actual sea Mayor que Set+Dur
        if (esValido) then
            esValido := satisfaceTimers(config, evMsg)
        endif

        ret_value := esValido

    end
```

Notar que se aplica a un evento la operación `getEvMsgHer()` y que devuelve el evento par correspondiente, asociado a la misma Etiqueta. Es decir que dada una Etiqueta *et*, el evento S(*et*) es evento hermano del evento R(*et*).

tipo BMSCMonitor

operación agregada

//Determina si un Evento temporizado cumple con las restricciones de tiempo definidas por los Timers activos //(Timers abiertos o recién expirados por Timeout), que existen sobre la instancia del Evento en la //configuración recibida

`satisfaceTimers?(in config: BMSCConfig, in ev: EventoTS) → ret_value: Bool`

Valida que para todo Timer abierto en *config*, *ev*.getTStamp() debe ser mayor o igual que el Timestamp asociado al evento Set del Timer y menor o igual que el Timestamp del Set del Timer más su duración.

Además valida que para todo Timer que expiró por Timeout en *config*, si *ev* es el evento inmediato siguiente, debe validarse que *ev*.getTStamp() sea mayor o igual que el Timestamp del evento Set del Timer más su duración.

```
satisfaceTimers?(in config: BMSCConfig, in ev: EventoTS)          → ret_value : Bool
    var
        evTimer          : EventoTS
        secEvTimers      : Secuencia_EventoTS
        esValido         : Bool
    begin

        esValido := true

        //Se obtiene la secuencia de EventoTS correspondientes a Timers de la instancia
        secEvTimers := config.getEventosDeTimerActivos(ev.getInstancia())

        //Se valida si existen EventoTS correspondientes a Timers para validar en la instancia del evento
        if (secEvTimers.size() > 0) then

            //Se recorre la secuencia de EventoTS correspondientes a Timers
            while (¬secEvTimers.vacia?() && esValido)

                //Se obtiene un EventoTS correspondiente a Timer de la instancia y se avanza el iterador
                evTimer := secEvTimers.getProximo()

                //Si el EventoDeTimer es un evento de "Set"
                if (evTimer.getTipo() == Set) then
                    //Se valida que el Timestamp del timer mas su duración,
                    //sea mayor o igual que el Timestamp asociado al evento
                    if (¬(ev.getTStamp() ≤ evTimer.getTStamp() + evTimer.Dur())) then
                        esValido := false
                        Error:  "Timestamp del evento no se encuentra en rango de un
                                 Timer abierto"
                    endif
                endif

                //Si el EventoDeTimer es un evento de "Timeout"
                if (evTimer.getTipo() == Timeout) then
                    //Se valida que el Timestamp del timer mas su duración,
                    //sea menor o igual que el Timestamp asociado al evento
                    if (¬(evTimer.getTStamp() + evTimer.Dur() ≤ ev.getTStamp())) then
                        esValido := false
                        Error: "Timestamp del evento menor que el Timestamp de Timeout
                                sucedido antes"
                    endif
                endif

            endwhile

        endif

        //Finalmente, ret_value tendrá el resultado si el evento cumple con las restricciones de tiempo de Timers
        ret_value := esValido
    end
```

tipo BMSCMonitor
operación agregada

//Determina si un Evento temporizado cumple con las restricciones de tiempo definidas por los
//Delay Intervals que posean al Evento como el segundo evento asociado, en la configuración recibida
`satisfaceDIs?(in config: BMSCConfig, in evMsg: EventoTS)` → `ret_value : Bool`
Valida que para todo Delay Interval (DI) d tal que ParEvDI(d) = (ev_1, *evMsg*),
evMsg.getTStamp() – ev_1.getTStamp(), se encuentren dentro del rango de tiempo definido por el DI.

```
satisfaceDIs?(in config: BMSCConfig, in evMsg: EventoTS)          → ret_value : Bool
    var
        di      : DelayInterval
        evUlt   : EventoTS
    begin

        ret_value := false

        //Si el evento tiene DI Mensaje asociado se sabe que es de Receive...
        if (evMsg.getDIMensaje() != null) then
            di := evMsg.getDIMensaje()

            //Se valida si se cumple la restricción del DI
            if (¬di.satisface(evMsg.getTStamp() - evMsg.getEvMsgHer().getTStamp())) then
                Error: "Timestamp no válido en DI de Mensaje"
            endif
        endif

        //Si el evento tiene DI de Velocidad de Procesamiento asociado...
        if (evMsg.getDIVelProc() != null) then
            di := evMsg.getDIVelProc()
            evUlt := config.getUltEvMsgMarcadoEnInst(evMsg.getInstancia())

            //Se valida si Timestamp de evMsg y del anterior satisfacen el DI
            if (¬di.satisface(evMsg.getTStamp() - evUlt.getTStamp()) then
                Error: "Timestamp no válido en DI Vel. Proc."
            endif
        endif

        //Sólo se ejecutará esta parte del método si se termina exitosamente
        //las pruebas sobre DIs VP y DIs MSG
        ret_value := true
    end
```

Notar que si un evento es el segundo evento asociado a un DI de mensaje o de velocidad de procesamiento, conoce del mismo a través de las operaciones `getDIMensaje()` y `getDIVelProc()`.

5 Implementación

5.1 Arquitectura

5.1.1 Arquitectura conceptual

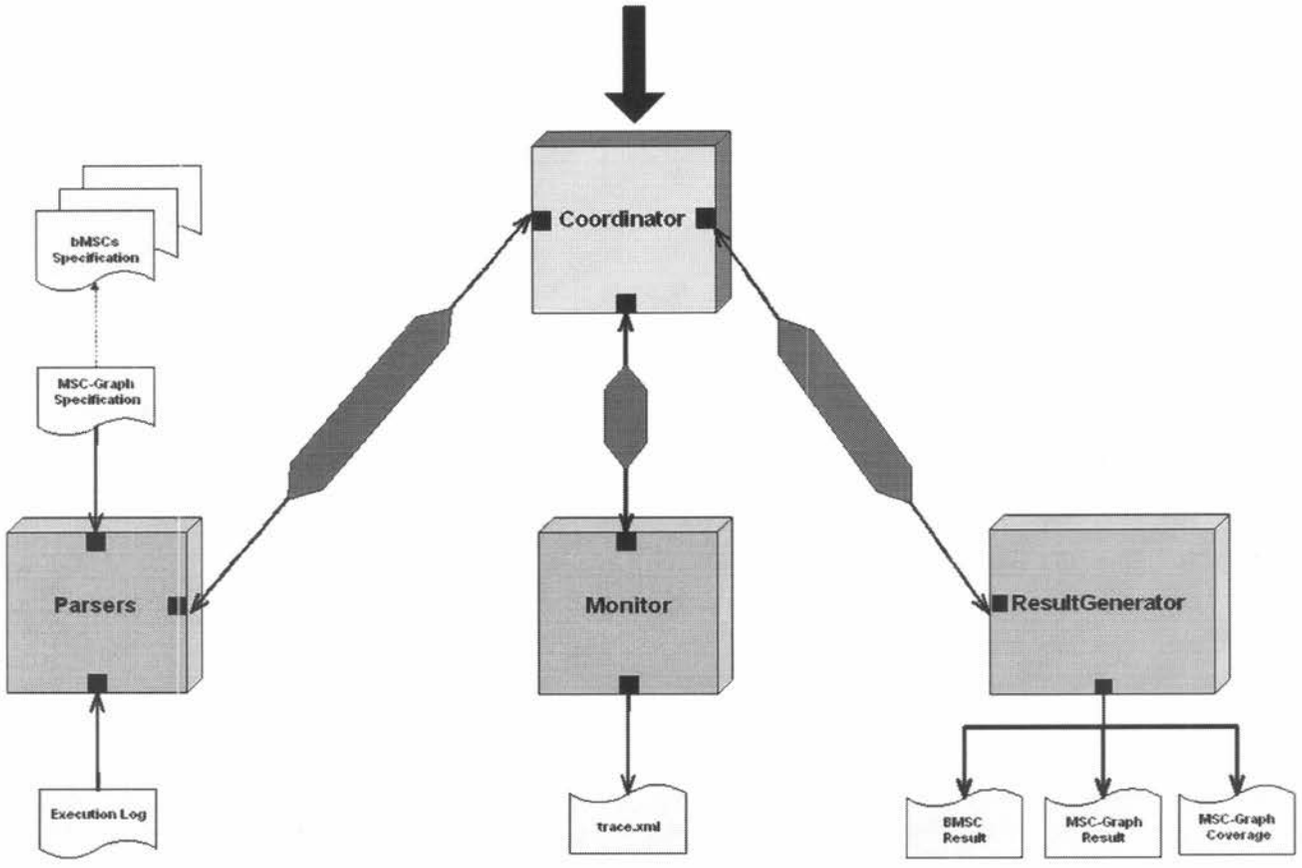


Figura 17

En la Figura 17 se visualiza el diagrama de la arquitectura conceptual donde se destacan cuatro módulos. El módulo de comienzo es el **Coordinator**, encargado de activar las tareas sobre los otros tres módulos. Él solicita a **Parsers** que interprete los archivos de entrada, luego solicita a **Monitor** que procese los datos recibidos y por último envía al **ResultGenerator** el resultado obtenido para que genere los archivos correspondientes.

El módulo **Parsers** se encarga de parsear los archivos de especificación MSC-Graph, bMSCs y logs de ejecución del sistema a monitorear.

El módulo **Monitor** procesa la entrada generada por **Parsers** y determina si el log de ejecución recibido es un prefijo válido del lenguaje denotado por la especificación recibida del sistema monitoreado. Durante el procesamiento el módulo genera un trace del mismo.

El módulo **ResultGenerator** recibe el resultado obtenido (los testigos activos) y genera los archivos de resultados.

5.1.2 Arquitectura de implementación

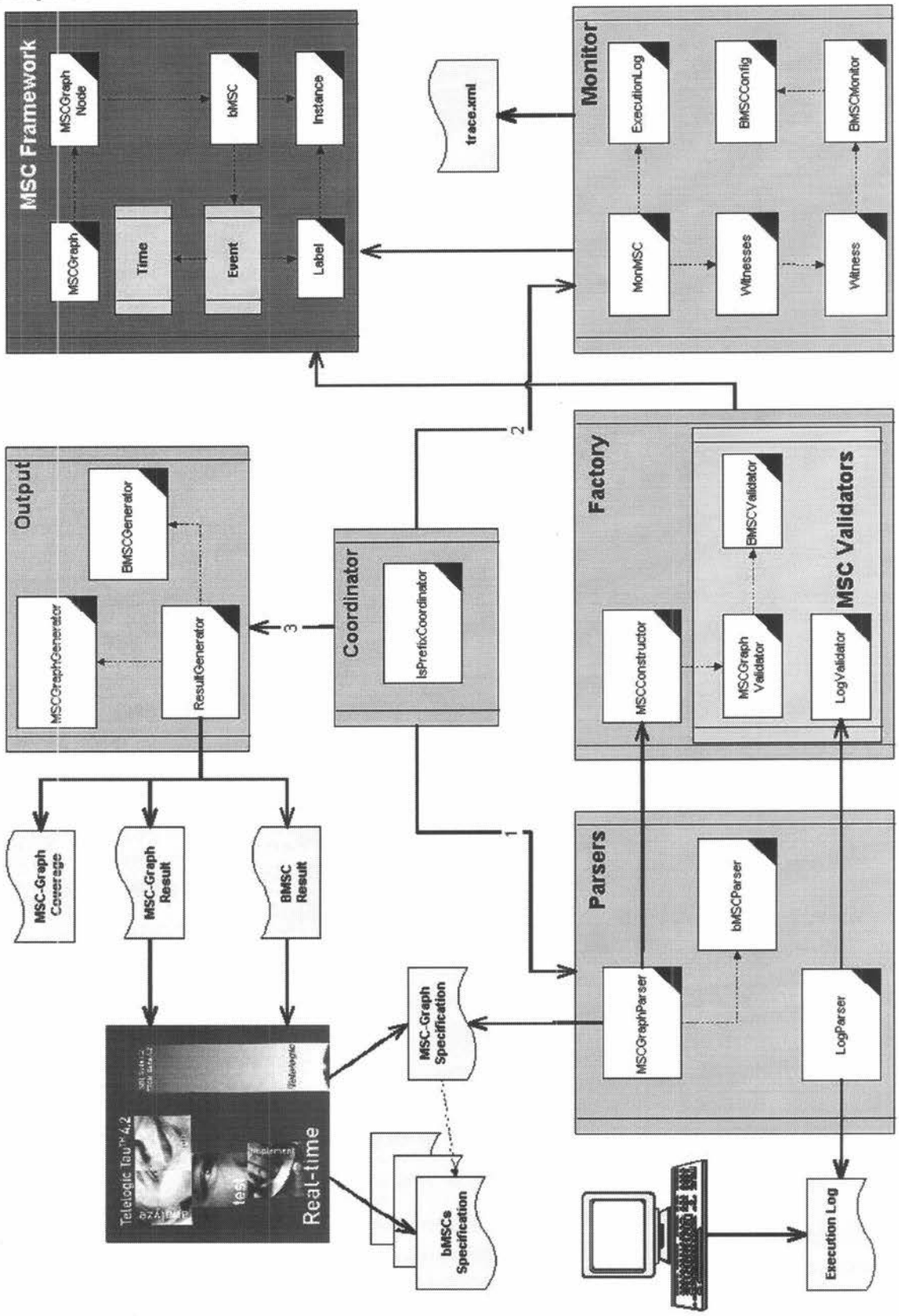


Figura 18

La arquitectura elegida para la implementación de la herramienta es una arquitectura modular. La solución propuesta es una herramienta separada en cinco componentes: los parsers, la validación y construcción de MSCs, el componente de monitoreo que consta del algoritmo principal, el componente de visualización del seguimiento y resultado obtenido, y el componente coordinador encargado de lanzar las tareas sobre cada uno de los anteriores. Todos los componentes utilizan un MSC Framework.

5.1.2.1 Entrada de datos

Como se puede observar en el diagrama de la Figura 18, se decidió utilizar para la generación de archivos de especificación MSC-Graph y bMSCs, la aplicación Telelogic Tau [TAU] (muy popular en el ambiente de las comunicaciones), dado que se consideró la más apropiada en el momento de elaboración de este trabajo.

La versión utilizada para el desarrollo de nuestra herramienta es la versión Telelogic Tau SDL Suite 4.2 (Copyright © 2001 Telelogic AB). La misma consta con un editor para MSC-Graphs y otro para bMSCs, donde se pueden generar las especificaciones en forma gráfica. A través de una opción del menú se permite generar el MSC-Graph y los bMSCs asociados a sus nodos, en su correspondiente sintaxis textual similar al estándar de recomendación Z.120.

A pesar que Telelogic Tau no respecta en un 100% la recomendación Z.120, la decisión de su utilización se debe a su fácil aprendizaje de uso y aunque no es totalmente compatible, lo es en mayor medida comparada con el resto de las herramientas analizadas.

El problema de ser 100% compatible con Z.120 se corresponde en algunos términos de tiempo. Uno de ellos es que para insertar una duración a los Timers, Tau agrega un prefijo delante de la misma. Otro problema es que no es posible crear Delay Intervals asociados a velocidad de procesamiento. Estos inconvenientes son solucionados con el agregado de "workarounds", que consisten en utilizar la aplicación Tau en una forma tal que se puede proveer la información necesaria sobre estos items en el momento de procesar la entrada.

En esta primera versión de la herramienta, la entrada de logging se toma a partir de archivos de texto, pero se espera que futuras versiones acepten la entrada a través de otros mecanismos como ser sockets TCP/IP, servicio de algún Middleware, etc.

5.1.2.2 Parsers

El primer componente de la herramienta es el de Parsers. Este encapsula la interpretación de los archivos de especificación generados por Telelogic Tau y genera estructuras intermedias con las líneas leídas de los archivos. Potencialmente podría reemplazarse esta sintaxis de entrada por otra diferente (XML por ejemplo).

Está compuesto por el BMSCParser, el MSCGraphParser y el LogParser, correspondientes a la lectura de bMSCs, MSC-Graphs y logs de ejecución respectivamente. El funcionamiento de los tres parsers es similar, validan la sintaxis de los archivos de manera top-down y a medida que la misma se determina correcta, genera las estructuras intermedias con los datos de las líneas parseadas.

5.1.2.3 Fábrica de MSCs (Factory)

El segundo componente de la herramienta es la fábrica de MSCs. El mismo recibe las estructuras intermedias generadas por el componente de parsers y se encarga de validar que semánticamente las mismas sean correctas para generar las estructuras del algoritmo principal.

Se compone de un MSCConstructor, que como su nombre lo indica es el constructor de MSCs, y de un subcomponente de validadores.

Este subcomponente de validadores está formado por el BMSCValidator, MSCGraphValidator y LogValidator, que se encargan de las validaciones a bMSCs, MSC-Graphs y logs de ejecución respectivamente. Realizan las validaciones semánticas que correspondan, es decir verifican la consistencia de los datos obtenidos para la generación en forma correcta de las estructuras principales.

Como los parsers sólo validan la sintaxis, no pueden asegurar que la semántica de los bMSCs y MSC-Graphs sea la correcta. Por lo tanto, luego de su ejecución, es necesario que se realicen ciertas validaciones a los datos obtenidos para verificar su consistencia.

Las validaciones que se realizan a los bMSCs luego de ser parseados son:

a) Consistencia entre las instancias declaradas en el bMSC y las que reciben ejecuciones de eventos. Una instancia no puede recibir una ejecución de un evento si no fue declarada, aunque sí es posible que se declare una y no reciba al menos una ejecución de evento (instancia Idle).

b) Consistencia de las instancias en las ejecuciones respectivas (Send y Receive) de las etiquetas. Es decir que tanto el envío como la recepción deben ejecutarse sobre el mismo par de instancias.
c) Consistencia de las instancias en las repeticiones de etiquetas. Es decir que si se repite la ejecución de una etiqueta, la misma debe suceder sobre el mismo par de instancias en el cual se ejecutó antes.

Las validaciones que se realizan al MSC-Graph luego de ser parseado son:

a) Existencia del nodo inicial.
b) Existencia de ids únicos de los nodos del MSC-Graph.
c) Consistencia de las instancias en las ejecuciones de las etiquetas de todos los bMSCs del MSC-Graph.
d) Existencia de ids únicos en los timers declarados en las distintas instancias de los bMSCs del MSC-Graph.

La validación que se realiza al log de ejecución luego de ser parseado es:

a) Existencia en el MSC-Graph de las etiquetas recibidas en el log.

Una vez validados los datos, el MSCConstructor recorre las estructuras intermedias generadas por el componente de parsers y genera las estructuras principales de la aplicación, haciendo uso de un framework, llamado MSC Framework.

5.1.2.4 Monitor

El componente de Monitoreo está compuesto por el algoritmo principal y las estructuras necesarias para realizar el monitoreo. Entre ellas se encuentran ExecutionLog (LogEjecucion), Witnesses (Testigos), Witness (Testigo), BMSCMonitor (monitor de un bMSC) y BMSCConfig (configuración de un bMSC).

Este componente también hace uso del MSC Framework y su principal función es validar que el log de ejecución del sistema recibido es un comportamiento válido de su especificación a través de MSC-Graph (y sus bMSCs respectivos). Se encarga de realizar las validaciones correspondientes a las restricciones de tiempo que se encuentran especificadas en los bMSCs del MSC-Graph.

Además brinda a través de una salida XML, los datos que se obtuvieron del seguimiento realizado y el resultado final de la ejecución de la herramienta. Como resultado final se obtienen los testigos activos correspondientes a la secuencia de etiquetas recibida.

5.1.2.5 MSC Framework

Este componente provee de todas las estructuras necesarias para que el componente de monitoreo pueda realizar la verificación del comportamiento del sistema que se está monitoreando.

Está conformado por MSCGraph y NodoMSCGraph para mantener la estructura de un MSC-Graph; y por BMSC (bMSC con restricciones de tiempo), Event (Evento), Label (Etiqueta), Instance (Instancia) y Time (Tiempo) para mantener la estructura de un bMSC.

Tanto Event como Time son subcomponentes. El primero de ellos contiene las estructuras necesarias para representar eventos de mensajes y de timers. El segundo las estructuras para representar timers y delay intervals.

5.1.2.6 Salida de datos (Output)

Output es el encargado de generar como salida un conjunto de archivos de texto. Está conformado por ResultGenerator, BMSCGenerator y MSCGraphGenerator. Por cada testigo activo, ResultGenerator genera un archivo MSC-Graph (por medio de MSCGraphGenerator) y un archivo bMSC (por medio de BMSCGenerator), representando respectivamente la secuencia de nodos recorridos de la especificación MSC-Graph original del sistema, y la ejecución de eventos correspondiente a la secuencia de etiquetas recibida a monitorear. Estos archivos se generan de acuerdo a la gramática definida por Z.120 para bMSCs y MSC-Graphs, con las modificaciones necesarias (debido a los problemas de algunas definiciones de restricciones de tiempo) para ser leídas por la aplicación Telelogic Tau.

Además genera un archivo XML con el cubrimiento sobre el MSC-Graph recibido. Este representa la unión de los cubrimientos de todos los testigos activos. El formato utilizado para este archivo es GXL (XML para grafos) [HW00].

5.1.2.7 Coordinador (Coordinator)

Finalmente, Coordinator es el encargado de pasarle a Parsers los archivos de log y de MSC-Graph a parsear. Luego para cada log de ejecución invoca al componente Monitor para que se ejecute la operación `esPrefijo?`. Por último, con cada resultado obtenido, invoca al componente Output para generar los archivos de resultado correspondientes.

5.2 Diseño y estructuras de representación

5.2.1 Diseño de la herramienta

A continuación se presentan los diagramas de diseño del MSC Framework y del módulo de Monitoreo, que representan la implementación de la solución teórica propuesta con anterioridad.

5.2.1.1 MSC Framework

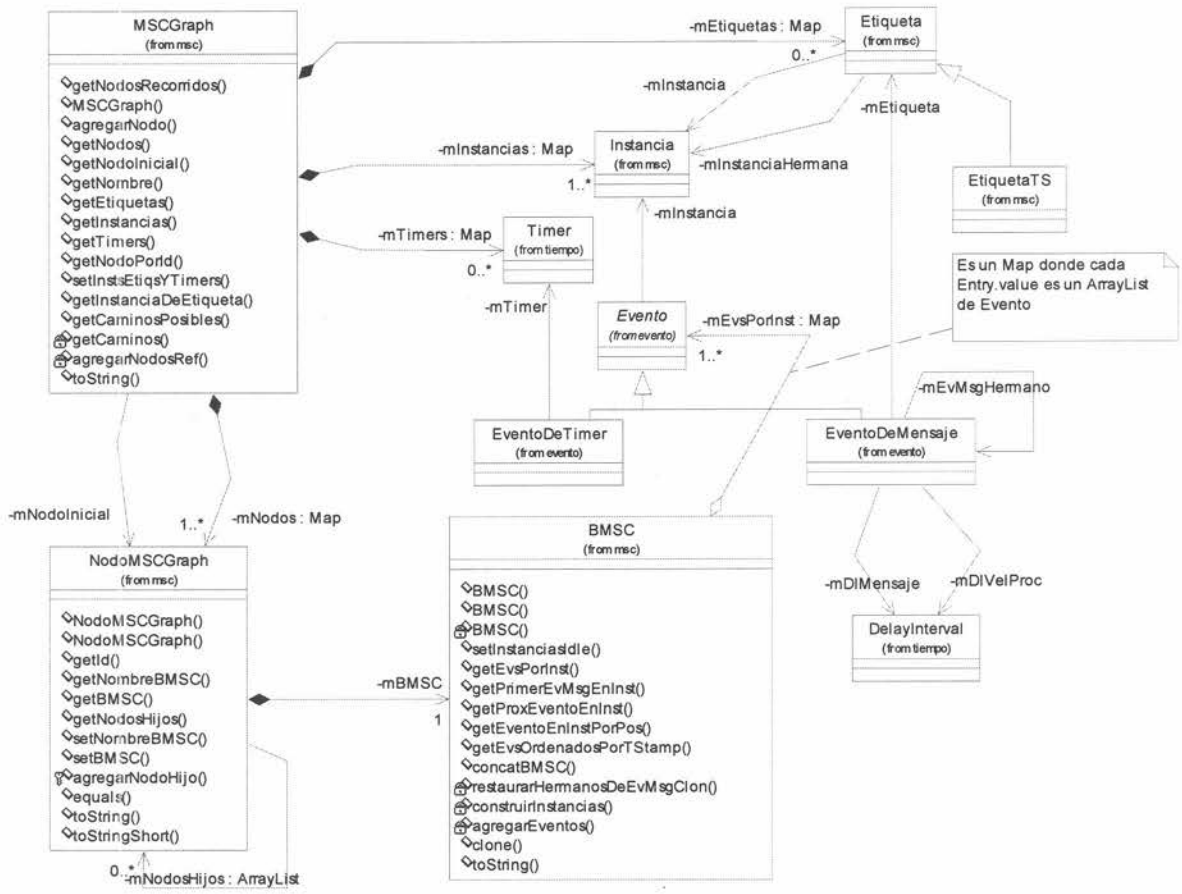


Figura 19

El MSC Framework, que se puede visualizar en la Figura 19, contiene aquellas clases necesarias para representar bMSCs, MSCGraphs y todas las definiciones enunciadas en la sección de sintaxis y semántica.

La clase **MSCGraph** posee métodos para establecer y obtener todos los componentes de un MSCGraph, así como métodos que el Monitor necesitará, como por ejemplo getCaminosPosibles. Para representar un MSCGraph, la clase se compone de varios Maps, uno de Etiquetas, otro de Instancias, otro de Timers y uno de NodoMSCGraph. Estos Maps son útiles a la hora de realizar validaciones a las restricciones de tiempo, ciclar por todas las instancias, o buscar una determinada etiqueta.

NodoMSCGraph representa un nodo del MSCGraph propiamente dicho y para mantener la estructura de MSC-Graph, tiene una lista de cuáles son sus hijos (mNodosHijos). Además, cada NodoMSCGraph se compone de un BMSC.

BMSC mantiene su estructura de Instancias y Eventos a través de un Map, donde cada Entry.Value es una Lista de Evento(s) ordenados (parcialmente). Aprovechando el polimorfismo, estos eventos pueden ser tanto de Mensaje, como de Timer, representados por EventoDeMensaje o EventoDeTimer respectivamente.

EventoDeTimer posee asociado un Timer, que determina la duración del mismo.

EventoDeMensaje debe saber quién es su hermano, y para ello apunta a otra instancia de EventoDeMensaje. Para saber qué tipo de EventoDeMensaje es (Send, Receive), se utilizan variables de instancia. Además, está asociado a una Etiqueta (cada EventoDeMensaje se puede asociar sólo a una etiqueta, pero una etiqueta puede tener múltiples EventoDeMensaje asociados).

Etiqueta conoce a qué instancia pertenece, así como la instancia a la que pertenece su etiqueta hermana, cosa necesaria durante el uso del Framework.

EtiquetaTS es una Etiqueta, con la diferencia que posee un Timestamp asociado, y se la utiliza para almacenar los datos obtenidos del log de ejecución.

5.2.1.2 Monitoreo

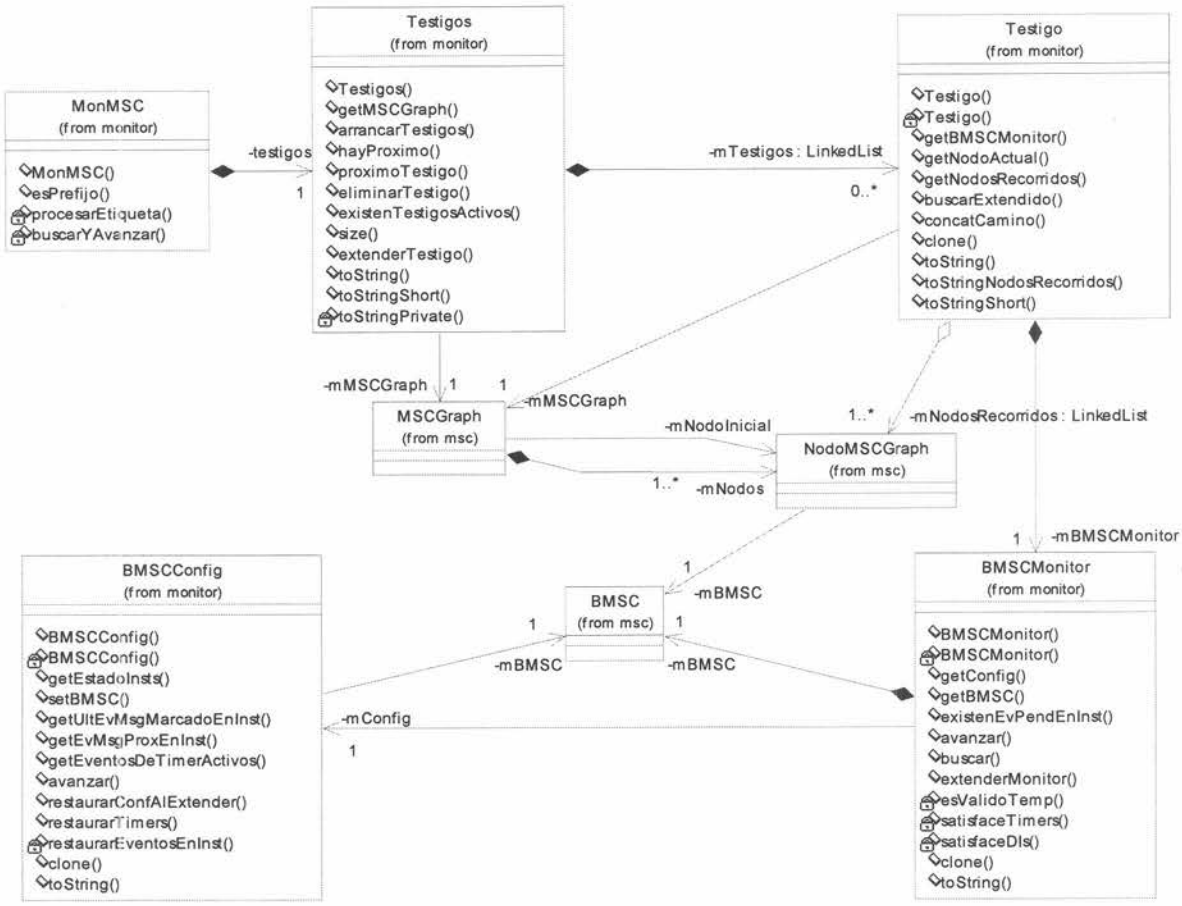


Figura 20

El diagrama de clases de monitoreo, que se puede visualizar en la Figura 20, presenta aquellas clases participantes en resolver el algoritmo `esPrefijo?`. La clase principal se llama MonMSC, que posee el método `esPrefijo?`, y es la que da comienzo al análisis desarrollado.

MonMSC es quien posee el algoritmo principal, explicado con detalle anteriormente. Se compone de un objeto Testigos.

Testigos posee todos los métodos para iterar a través de los testigos activos, y quien a su vez se compone de una lista de Testigo.

Testigo mantiene por un lado una lista de nodos recorridos en mNodosRecorridos, y por otro lado se compone de un BMSCMonitor.

BMSCMonitor es un CaTSE. Mantiene un BMSC subyacente (mBMSC), creado en forma dinámica y que es el resultado de la concatenación de los nodos recorridos por el Testigo al que pertenece. Permite buscar eventos válidos dentro del bMSC subyacente. Además, posee una configuración (mConfig), que representa el estado en que se encuentra el bMSC subyacente (porción del bMSC cuyos eventos ya fueron ejecutados).

BMSCConfig representa una configuración de un BMSCMonitor y conoce al mismo bMSC que contiene BMSCMonitor. Posee métodos para mantener la consistencia en los próximos posibles eventos a ocurrir por instancia, en los ya ocurridos en cada una de ellas y para poder avanzar sobre los eventos.

5.2.2 Estructuras de representación

A continuación se describen las estructuras de representación de los tipos utilizados en la solución 4.5 Monitoreo de MSC-Graphs, que no se describieron con anterioridad porque no eran necesarias para comprender los algoritmos.

Testigos == tupla < *mTestigosAct* : Secuencia_Testigo , *mMSCGraph* : MSCGraph >

El Tipo *Testigos* posee una Secuencia de Testigos "activos", es decir, aquellos testigo(s) que representan ser CaTSEs de la secuencia recorrida hasta el momento.

Notar que es posible mejorar la forma de representar los testigos activos, para optimizar por ejemplo espacio de almacenamiento. Se podría utilizar otra estructura en lugar de una secuencia, por ejemplo un árbol, tal que diferentes testigos activos, posean similares "ramas" como prefijos en común. Esto es parte de los trabajos propuestos en "8 Trabajos a futuro".

Adicionalmente, *Testigos* referencia al MSCGraph sobre el cual se está realizando el monitoreo.

El Inviariante de representación determina que todo *Testigo* que es parte de la secuencia *mTestigosAct*, referencia en todos sus componentes al mismo MSCGraph que *Testigos*.

Testigo == tupla < *mNodosRecorridos* : Secuencia_NodoMSCGraph ,
mBMSCMonitor : BMSCMonitor ,
mMSCGraph : MSCGraph >

El Tipo *Testigo* se compone de *mNodosRecorridos*, que determina cuál fue el camino en el *mMSCGraph* que el *Testigo* recorrió hasta el momento, y permite mantener un seguimiento de los nodos recorridos, así como generar el resultado en el momento de la salida.

Además, posee un *mBMSCMonitor* que es quien se encarga de monitorear el BMSC dinámicamente generado, y que contiene la concatenación de todos los nodos recorridos determinados por *mNodosRecorridos*.

Finalmente, posee *mMSCGraph*, que referencia al MSCGraph con el que se trabaja.

El invariante de representación determina que *mNodosRecorridos* posee nodos de *mMSCGraph* y que el BMSC de *mBMSCMonitor* es producto de concatenar todos los nodos de *mNodosRecorridos*.

BMSCMonitor == tupla < *mBMSC* : BMSC , *mConfig* : BMSCConfig , *mSeqRec* : Secuencia_EtiquetaTS >

El Tipo *BMSCMonitor* posee una tupla de dos componentes. El primero de ellos, *mBMSC*, representa un BMSC que se genera en forma dinámica, y será el resultado de concatenaciones de NodosMSCGraph.

El segundo componente, *mConfig*, representa la configuración que posee *mBMSC*, y posee toda la información necesaria para saber el estado de avance en que se encuentra *mBMSC* .

El tercer componente, *mSeqRec*, representa la secuencia de etiquetas temporizada reconocida por el bMSC que denota *mConfig*.

El invariante de representación determina que *mConfig* es una configuración válida de *mBMSC* y *mSeqRec* $\in$ $Lin_p(M)$ donde M es el bMSC que denota *mConfig*.

5.3 Métricas de la herramienta MonMSC

A continuación se muestran resultados sobre mediciones realizadas a las clases implementadas de la herramienta MonMSC. Las mismas expresan diversas mediciones como ser la cantidad de clases utilizadas, cantidad de clases creadas, líneas de código, líneas de comentarios, etc. que permiten obtener una idea cuantitativa del tamaño de la herramienta.

Item	Valor
Clases utilizadas	152
Clases creadas	**76**
Métodos	924
Líneas	17114
Líneas en Blanco	2196
Líneas de Código	**10768**
Líneas de Comentarios	**4168**
Sentencias ejecutables	6525
Sentencias declarativas	2199
% de Comentarios	**38%**

Por ejemplo, se puede observar que son 76 las clases creadas para la herramienta y que el porcentaje de comentarios sobre la cantidad de líneas de código resulta ser del 38% (4168 líneas de comentarios sobre las 10768 líneas de código).

5.4 Manual de uso de la herramienta MonMSC

5.4.1 Instalación de MonMSC

Prerrequisitos:

1. JRE o JDK SE 1.4 o superior (www.java.sun.com/j2se)

Pasos de instalación:

1. Ejecutar el archivo Install.bat que se encuentra en el directorio `tool` en el material entregado.
   Sintaxis: `Install c:\MonMSC` donde `c:\MonMSC` es el directorio donde se instalará la herramienta.

5.4.2 Modo de uso de MonMSC

Para invocar la herramienta MonMSC es necesario escribir:

```
java -jar MonMSC.jar -M <MSCGraphFile> -L <LogFileOrDir> [-O <OutputPath>]
 [-D <DebugLevel>] [-U <TimeUnit>] [-TA | -TP]
```

donde cada parámetro representa :

```
-M <MSCGraphFile>   Archivo .mpr que contiene el MSCGraph base a utilizar.
                    Este debe estar en formato Telelogic TAU, y referenciar a cada
                    uno de los archivos bMSCs (que deben estar en el mismo
                    directorio que el archivo del MSC-Graph).
                    Parámetro obligatorio. No hay default.

-L <LogFileOrDir>   Puede representar un archivo de Log, o un directorio desde donde
                    obtener todos los archivos de log.
                    En caso de ser sólo un archivo, se espera el nombre del mismo.
                    En caso de ser un directorio, se espera un Path tal que se
                    obtendrán todos los archivos con extensión .log (o la que se
                    especifique en el archivo de configuración).
                    Parámetro obligatorio. No hay default.

-O <OutputPath>     Path donde se grabarán archivos resultantes.
                    Parámetro optativo. Default: dirActual\output
                    Nota: En caso que el directorio no se especifique, y
                    dirActual\output no exista, este será creado.
-D <DebugLevel>     Nivel de salida en el archivo de trace que se desea, el mismo
                    es incremental, es decir, los números más grandes incluyen a
                    los mensajes de números máss chicos:
                    0=Sin Salida: No se imprime nada en los archivos de Trace.
                    1=Errores   : En caso que falle el algoritmo esPrefijo, se
                                  imprimirán los errores obtenidos únicamente.
                    2=Advertenc.: Se incluyen mensajes de error y Advertencias.
                    3=Statuses  : Se incluyen también mensajes de status que
                                  sucedan durante la ejecución
                    4=Todo      : Se imprime cualquier mensaje que la herramienta
                                  genera.
                    Parámetro optativo. Default: 3

-U <TimeUnit>       Unidad en la que se encuentran representadas las restricciones
                    de tiempo en los archivos de entrada:
                    mili  : milisegundos
                    seg   : segundos
                    min   : minutos
                    Parámetro optativo. Default: mili
                    Nota: los bMSCs resultantes siempre estarán expresados en mili
-TA | -TP           Permite elegir si se asocia el Timestamp de los Eventos de
                    Timer al Timestamp del Evento de Mensaje anterior (-TA) o al
                    posterior (-TP)
                    Parámetro optativo. Default: TA
                    Nota: Si se elige -TA, el Timestamp de aquellos Eventos de
                    timer minimales sera el del evento posterior.
```

5.4.2.1 Lectura de parámetros

La herramienta está preparada para recibir parámetros de uso a través de línea de comando como recién fue explicado, o a través de un archivo de configuración.

El archivo de configuración tiene el nombre `MonMSC.properties`, y permite establecer los siguientes parámetros:
Nota: Este archivo debe encontrarse en el mismo directorio que MonMSC.jar

```
OutputPath=<PathDeDirectorioDeSalida. Ej: c:\output>
DebugLevel=<0 | 1 | 2 | 3 | 4>
UnidadTiempo=<mili | seg | mim>
AsocTStampEvTimer=<A | P>
ExtensionLOG=<ExtensionDeArchivosLog. Ej: log>
ExtensionMSC=<ExtensionDeArchivosMSC. Ej: mpr>
```

donde

OutputPath	Similar a "-O <OutputPath>"
DebugLevel	Similar a "-D <DebugLevel>"
UnidadTiempo	Similar a "-U <TimeUnit>"
AsocTStampEvTimer	A : Similar a "-TA" P : Similar a "-TP"
ExtensionLOG	Extensión que se utilizará para buscar todos los archivos de Log que se encuentren en el directorio recibido a través de "-L <LogFileOrDir>"
ExtensionMSC	Extensión que se utilizará para generar los archivos de bMSC y MSC-Graph que se obtengan como resultado

El orden de prioridades de uso para los parámetros es:

1. Si se recibe por línea de comandos, es el valor que se toma.
2. Si no existe por línea de comandos, se busca en el archivo de configuración.
3. Si no existe en el archivo de configuración, se utiliza el valor por default de la herramienta.

5.4.3 Modo de uso de Telelogic TAU

Aquí se describe cómo se debe utilizar la herramienta Telelogic Tau para la entrada y salida de datos.

Se la utilizará como entrada, para generar una especificación MSC-Graph, con sus bMSCs respectivos, de forma válida para ser interpretada correctamente por MonMSC.

Se la utilizará como salida, para visualizar el bMSC que valida la ejecución recibida en el log como un comportamiento correcto y especificado.

5.4.3.1 Entrada de datos

MSC-Graphs

Para construir MSC-Graphs se debe seleccionar el editor de HMSCs desde el menú **Tools**, submenú **Editors** y allí la opción **HMSC Editor**. De esta forma, se presenta el editor junto con una barra de símbolos que permitirán construir un MSC-Graph.

La barra de símbolos se describe a continuación:

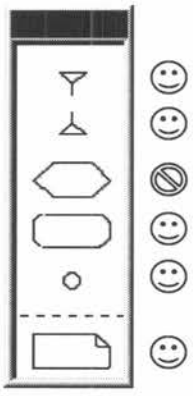


Figura 21

1. Nodo inicial de un MSC-Graph y representa el comienzo del mismo.
2. Nodo final de un MSC-Graph y representa el fin del mismo.
3. Condición: representa una condición/estado en el que se encuentra un MSC-Graph.
4. Nodo del MSC-Graph y referencia a un bMSC asociado.
5. Punto de conexión, utilizado habitualmente cuando desde un nodo hay más de una conexión con otros nodos.
6. Texto, utilizado para escribir comentarios globales en el MSC-Graph.

De todos los símbolos de la Figura 21, se podrán utilizar para generar una especificación MSC-Graph válida, el símbolo del nodo inicial (comienzo) (1) , el símbolo del nodo final (2), el de nodo (bMSC asociado) (4), el punto de conexión (5) y el texto de comentarios (6). Es decir, queda excluido el símbolo de condición.

De los símbolos permitidos, es requisito obligatorio para crear un MSC-Graph cuyo archivo correspondiente a su sintaxis textual sea válido, utilizar el nodo inicial, el nodo final y el nodo que referencia un bMSC. El punto de conexión y el texto para comentarios son optativos.

Para generar el archivo de especificación compatible con la sintaxis concreta definida, se debe seleccionar en el menú **Tools**, el submenú **Generate MSC PR**.

Un ejemplo utilizando los símbolos permitidos puede verse a continuación:

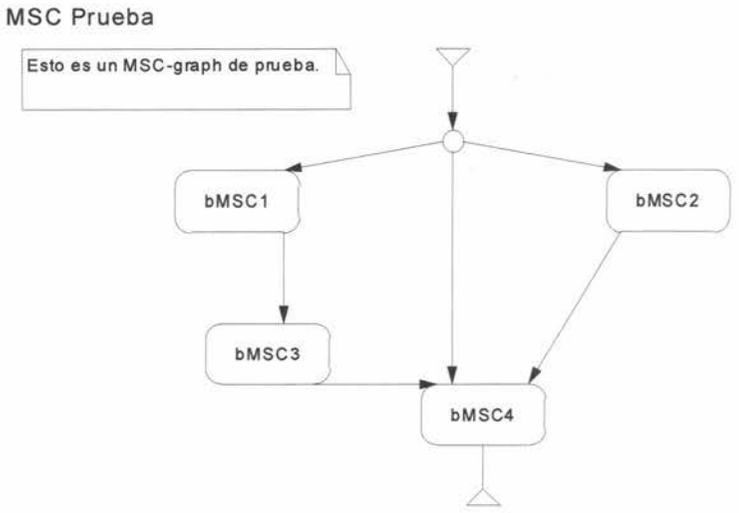


Figura 22

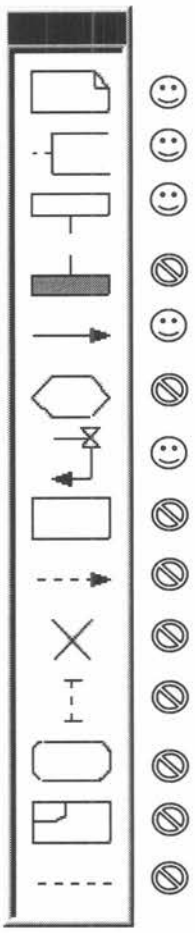
En el MSC-Graph de la Figura 22 se observa un nodo inicial conectado a un punto de conexión. A través de éste, el nodo inicial puede conectarse con los nodos asociados a bMSC1, bMSC2 y bMSC4. El nodo bMSC1 se conecta con el nodo bMSC3. El nodo bMSC3, como así también el bMSC2, se conectan al bMSC4. El nodo bMSC4 se conecta con el nodo final del MSC-Graph. También se observa un comentario global con el texto "Esto es un MSC-graph de prueba.".

Según la sintaxis propuesta en la sección 3.2.3 Sintaxis de un MSC-Graph, un MSC-Graph posee un conjunto de nodos formado por dos subconjuntos: uno con un único nodo inicial y otro por nodos intermedios; por lo tanto el nodo final que se visualiza en la Figura 22 no es interpretado por la herramienta MonMSC. Debido a la decisión de utilizar la herramienta Telelogic Tau para la creación de los MSC-Graphs, es necesaria su inclusión, pero como no tiene utilización alguna, no se lo interpreta.

bMSCs

Para construir bMSCs se debe seleccionar el editor de bMSCs desde el menú **Tools**, submenú **Editors** y allí la opción **MSC Editor**. De esta forma, se presenta el editor junto con una barra de símbolos que permitirán construir un bMSC.

La barra de símbolos se describe a continuación:



1. Texto para comentarios globales en el bMSC.
2. Comentarios a nivel evento
3. cabeza de una instancia
4. Cola de una instancia y determina el fin de la descripción de la instancia en el bMSC.
5. Mensaje.
6. Condición, representa una condición/estado en el que se encuentra un bMSC.
7. Timer, permite eventos de Set con su duración, Reset y Timeout.
8. Ejecución de alguna acción.
9. Creación dinámica de instancias.
10. Fin de una instancia.
11. Corregión (zona donde los eventos pueden ejecutarse en forma desordenada).
12. Referencia a un bMSC.
13. Expresión inline (alternativas, ejecuciones paralelas, ciclos, etc .).
14. Separador de una expresión inline.

Figura 23

De todos los símbolos de la Figura 23, se podrán utilizar para generar un bMSC válido, el símbolo de texto para comentarios globales (1), el de comentarios a nivel evento (2), la cabeza de una instancia (3), el de mensaje (5) y el de timer(7). Es decir, quedan excluidos los siguientes símbolos: la cola de una instancia, condición, acción, creación dinámica de instancias, fin de una instancia, corregión, referencia a bMSC, expresión inline y separador de expresión inline.

De los símbolos permitidos, es requisito obligatorio para crear un bMSC cuyo archivo correspondiente a su sintaxis textual sea válida, utilizar solamente instancias. Los mensajes, comentarios globales, comentarios a nivel evento y timers son optativos.

Para generar el archivo de especificación compatible con la sintaxis concreta definida, se debe seleccionar en el menú **Tools**, el submenú **Generate MSC PR**.

Se presenta como ejemplo en la Figura 24, dos bMSCs del caso ATM utilizando los símbolos permitidos.

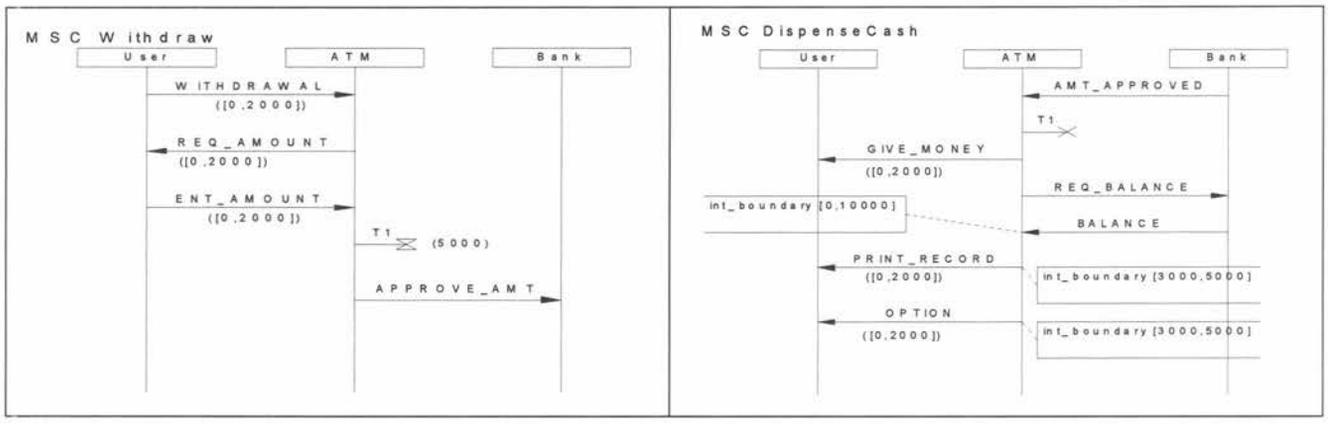


Figura 24

En los bMSCs de la Figura 24 se observan tres instancias: User, ATM y Bank. Se pueden observar varios mensajes intercambiados por las instancias: WITHDRAWAL desde User hacia ATM, REQ_AMOUNT desde ATM hacia User, AMT_APPROVED desde Bank hacia ATM, etc.

En la instancia ATM del bMSC *Withdraw* existe el evento Set del Timer T1 con duración 5000 unidades, antes del envío del mensaje APPROVE_AMT y en el bMSC *DispenseCash* existe el evento de Reset correspondiente.

Todos los mensajes entre las instancias User y ATM tienen asociado un delay interval cuyo intervalo de tiempo es [0, 2000] unidades. Estos delay intervals se deben especificar en el campo utilizado para los parámetros del mensaje, debido a que Telelogic TAU no permite crearlos.

A su vez se pueden observar tres delay intervals asociados a velocidad de procesamiento: uno sobre el evento recepción del mensaje BALANCE con el intervalo [0, 10000] que lo asocia con el envío del mensaje REQ_BALANCE en la instancia ATM, otro sobre el evento envío de PRINT_RECORD con el intervalo [3000, 5000] que lo asocia con la recepción del mensaje BALANCE en ATM, y el último sobre el evento de envío OPTION con otro intervalo [3000, 5000] que lo asocia con el envío del mensaje PRINT_RECORD también en ATM.

Como puede verse, el uso de comentarios a nivel evento es utilizado **para definir delay intervals asociados a velocidad de procesamiento**. Siempre debe asociarse al segundo evento correspondiente al delay interval y anteponiendo la palabra "**int_boundary**" al intervalo de tiempo. Esta definición de uso, se debe a que la herramienta Telelogic TAU no permite definir este tipo de delay intervals.

No se permite utilizar el símbolo de comentario a nivel evento para escribir comentarios, sólo debe usarse para definir delay intervals de velocidad de procesamiento. Para escribir comentarios se debe usar el símbolo de texto para comentarios globales.

Para generar el archivo de especificación compatible con la sintaxis concreta definida, se debe seleccionar en el menú **Tools**, el submenú **Generate MSC PR**.

5.4.3.2 Salida de datos

Para interpretar en forma correcta el bMSC generado por la herramienta MonMSC, se describe con el ejemplo de la Figura 25.

En el bMSC se pueden observar los eventos que matchean con cada etiqueta recibida en el log de ejecución, es decir la especificación que verifica el log como correcto. Existen comentarios sobre los eventos de envío y recepción de mensaje representando el Timestamp en el cual ocurrió cada uno. Este Timestamp se corresponde con el que tiene asociado la etiqueta en el log de ejecución y que matchea con el evento.

Se aclara que sólo se incluyen aquellos eventos que hayan ocurrido de acuerdo al log recibido (los que matchean con sus etiquetas). Es decir que si existen eventos que no fueron procesados, no se incluirán.

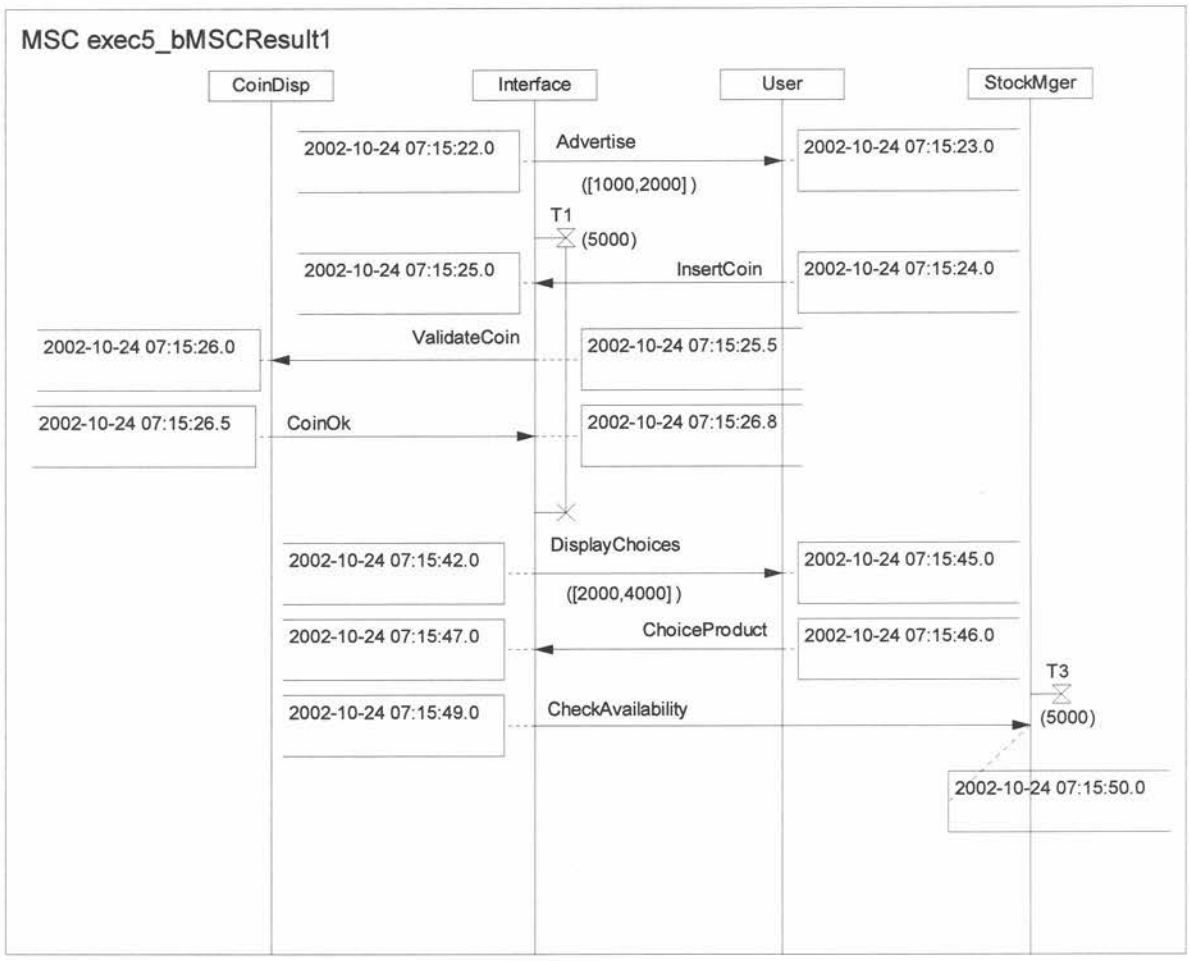


Figura 25

Notas de la implementación de MonMSC:

No es obligatorio que todos los bMSCs asociados a los nodos de un MSC-Graph tengan las mismas instancias. Por más que en la definición de la sintaxis abstracta de un MSC-Graph se pida eso, no se restringe en el momento de la creación de los bMSC en una especificación que se la utilice con la herramienta MonMSC. Como en el desarrollo del trabajo se asume existencia de todas las instancias en todos los bMSCs, se utiliza la unión de las instancias de todos los bMSCs, y así será representada la salida.

No se registran las instancias que son Idle en todo el MSC-Graph, es decir aquellas instancias que son Idle en todos los bMSCs del MSC-Graph.

Todos los DIs, ya sean de velocidad de procesamiento o de mensaje, deben tener definidos sus intervalos completamente abiertos "(,)", o completamente cerrados "[,]". Esto es así ya que en la versión actual de TAU, no se permite un intervalo Semi-Abierto (se subraya como si fuese un error sintáctico).

5.5 Metodología de desarrollo

Para el desarrollo de este trabajo se han considerado diferentes aspectos del proceso de desarrollo de Software, que consideramos importantes para lograr un resultado exitoso en la construcción de una herramienta que resuelva el problema planteado.

5.5.1 Diseño e implementación

Luego de resolver formalmente el problema, se comenzó a elaborar la herramienta MonMSC.

El primer paso fue el de diseñar un Framework de clases de MSCs, que provea las operaciones mínimas necesarias para lo que hasta ese momento se necesitaba. Para armar este Framework de clases y esqueletos de métodos, se utilizó la herramienta *Rational Rose* (www.rational.com), a partir de la cuál se generó código Java, que fue la base de la implementación.

Una vez obtenido el código inicial, se comenzó a implementar en el lenguaje *Java (*www.java.sun.com), utilizando la herramienta *Borland JBuilder 7* (www.borland.com/jbuilder). Esta herramienta provee numerosas funcionalidades en lo que respecta a manejo de código fuente, referencias a clases, diagramas UML, refactoring, integración con otras herramientas, etc., y fue la que nos acompañó a lo largo de toda la implementación.

Para generar los parsers que implementan las sintaxis concretas de bMSCs, MSC-Graphs y Logs de ejecución, se utilizó *JavaCC 2.1* (www.metamata.com/javacc), herramienta que facilita el diseño y la implementación de lenguajes propios en Java. *JavaCC* es un generador de parsers que permite definir gramáticas similares a la forma Backus-Naur de manera muy sencilla (notación alto nivel).

En ciertas partes de la implementación, y cuando consideramos que la complejidad del código era importante, utilizamos la técnica de Pair Programming (www.extremeprogramming.org/rules/pair.html) para asegurar mejor calidad y menor cantidad de defectos sobre el mismo.

Una vez terminado un primer Release estable y con todas las funcionalidades requeridas, se procedió a realizar Reverse Engineering, utilizando *Rational Rose* para generar diagramas de clases que representen las distintas relaciones existentes entre las mismas.

En la sección 5.2 Diseño se pueden visualizar los diagramas de diseño resultantes del MSC Framework y el módulo de Monitoreo.

5.5.2 Configuration Management y administración de Artifacts

Para poder trabajar concurrentemente sobre el proyecto, se ha establecido un repositorio centralizado de Artifacts (código fuente, documentos, archivos de configuración, imágenes, etc.), utilizando como herramienta *CVS* (www.cvs.org). La interfaz de uso desde Windows para *CVS* utilizada fue *WinCVS* (www.wincvs.org).

Para lograr acceso remoto, se utilizó comunicación encriptada sobre Internet, utilizando el producto *Putty* (www.chiark.greenend.org.uk/~sgtatham/putty/), que permitió a través de telnet SSH, acceder al puerto que *CVS* requiere. Cabe notar que el server residió en la casa de uno de los integrantes de este trabajo, con una conexión ADSL.

Así, se logró trabajar en una forma geográficamente distribuida, utilizando la técnica de Updates y Commits.

Como funcionalidad agregada, cada commit realizado, dispara un mail a los integrantes del proyecto, informando los comentarios que se han ingresado en el commit, y permitiendo una comunicación y control mayor en el proyecto.

Para lograr un manejo ordenado de Artifacts, se generaron numerosos Baselines (Tags en *CVS*), los cuales permitieron darnos una noción de avance sobre el desarrollo, y la situación en que se encontraba el proyecto en todo momento. Para esto, cada Baseline se encuentra acompañado de un Release Notes, donde se incluyen comentarios sobre las nuevas funcionalidades a partir de Releases anteriores.

Una vez terminadas las primeras versiones de implementación, se comenzó a realizar el deployment de la aplicación. Para esto, se utilizó la herramienta *Ant* (www.jakarta.apache.org/ant), siendo ésta un Make multiplataforma, que nos permitió generar un archivo .jar con las configuraciones necesarias para invocar la herramienta MonMSC por línea de comando.

5.5.3 Testing

Durante todo el proceso de implementación en Java, se ha utilizado la herramienta *JUnit* (www.junit.org), que nos ha permitido realizar tests automatizados de unidad, tests de regresión y tests funcionales. Esta herramienta posee una integración con *JBuilder*, permitiendo que desde el ambiente de desarrollo se lance la ejecución de los tests.

Para utilizar *JUnit*, se ha creado un package "test" en el código, que posee todos los casos de test a verificar. *JUnit* provee una clase TestCase, que es necesario heredarla para generar los casos de test propios.

Adicionalmente, es posible crear un Test Suite, que básicamente permite ejecutar múltiples TestCases en una misma corrida.

Cada clase de test propia representa uno o muchos tests realizados. Cada una de estas clases puede tener múltiples métodos tal que su nombre comience con el prefijo "test". Estos métodos son ejecutados por la herramienta *JUnit*, llamando previamente al *setUp*() de la clase para establecer el ambiente inicial, y luego al *tearDown*() para terminar el caso de test.

Dentro de los métodos creados, se escriben **aserciones**, de forma que si una aserción no es cumplida, el caso de test se considera **no exitoso**. Si todas las aserciones de un caso son correctamente evaluadas, el caso de test resulta **exitoso**. De esta forma, si todos los casos de test de una clase son exitosos, el test de la clase también lo es.

A continuación se enumeran las clases de Test creadas:

Clase	Funcionalidad
`AllTests`	Test Suite que permite ejecutar todos los tests definidos.
`TestATMCase`	Caso de test del ejemplo ATM descripto en la sección 6 Casos de Estudio. Posee dos métodos de test: 1) testEstructuraMSCGraph: verifica que la estructura MSCGraph obtenida a partir del parsing de los archivos de entrada sea correcta. 2) testNodoDispenseCash: verifica que la estructura del nodo DispenseCash (uno de todos los nodos del Grafo) sea correcta a partir de la entrada.
`TestBMSCConfig`	Caso de test que valida diferentes funcionalidades de la clase BMSCConfig. Como esta clase se considera con funcionalidades importantes, se crearon métodos de test sobre ella.
`TestEquals`	Valida las comparaciones entre diferentes clases implementadas, para asegurarse que la igualdad se comporta en forma correcta. Se incluyen entre otras Eventos de Timer y de Mensaje, Instancias, Etiquetas, DIs, Nodos, etc.
`TestLogPreProcessor`	Valida que el parsing de algunos casos de ejemplo correctos de Log de ejecuciones se realiza correctamente y que los casos de error efectivamente son erróneos.
`TestMSCGraphParser`	Valida que el parsing de algunos casos de ejemplo correctos de MSCGraphs se realiza correctamente y que los casos de error efectivamente son erróneos.
`TestMultTesCase`	Caso de test de un ejemplo que permite tener múltiples testigos activos. Posee diversas ejecuciones del algoritmo principal `esPrefijo?`, verificando la salida del mismo. Para la verificación se utiliza el *ValidadorDeTests*. Adicionalmente, verifica que la estructura obtenida desde el Parsing sea correcta.
`TestVMCase`	Caso de test del ejemplo VM descripto en la sección 6 Casos de Estudio. Posee diversas ejecuciones del algoritmo principal `esPrefijo?`, verificando la salida del mismo. Para la verificación se utiliza el *ValidadorDeTests*. Adicionalmente, verifica que la estructura obtenida desde el Parsing sea correcta.
`ValidadorDeTests`	Métodos utilizados por diversos casos de test que requieran validar resultados del método `esPrefijo?`. Verifica diversos puntos sobre el resultado: cantidad de testigos obtenidos, configuración de los mismos, toString() de los Testigos, nodos recorridos, etc.

Cada vez que se agrega nueva funcionalidad, o se modifica alguna parte del código, se ejecutan nuevamente estos tests de *JUnit*, permitiendo tener una especie de oráculo sobre lo que se espera obtener en cada caso de test y ver si efectivamente ha ocurrido lo esperado.

En lo que respecta a Testing funcional, se realizaron numerosos casos de test que cubren diferentes ramas de ejecución, y que permitieron detectar y corregir bugs sobre la evaluación de estos casos. Los tests `TestMultTesCase` y `TestVMCase` enunciados en la tabla anterior, poseen casos que cubren el espectro de posibilidades funcionales que

resultan de ejecutar el algoritmo `esPrefijo?`. También para cada ejecución de test funcional, se verifica que el resultado obtenido sea de éxito o fracaso, según lo esperado.

5.5.4 Javadoc

El código fuente fue cuidadosamente documentado, especialmente lo que respecta a métodos públicos, clases y variables de instancia.

Para el formato de la documentación se utilizó el Javadoc, siendo el estándar en documentación de código Java. Se acompaña al presente trabajo el *Javadoc* de la tesis, que se podrá encontrar junto con el material entregado en el directorio `src`.

6 Casos de Estudio

Para ilustrar el análisis del presente trabajo, se presentan a continuación dos casos de estudio que permitirán al lector lograr una mejor comprensión de la herramienta. El primero de ellos es el caso del ATM que se presentó en la introducción y el segundo una máquina expendedora de productos (VM). Los mismos fueron generados con la herramienta Telelogic Tau.

En los seguimientos de los ejemplos que se describirán en las siguientes secciones, se asume que los Timestamps de los eventos de Timers, se asocian a los Timestamps de los eventos de mensaje **anteriores**, y que las restricciones de tiempo se expresan en **milisegundos**.

6.1 Cajero automático (ATM)

Como primer caso de estudio se recuerda la especificación MSC del cajero automático, de ahora en más llamado ATM, y que se puede visualizar en las Figura 1, Figura 26y Figura 27. La primera figura representa la especificación del comportamiento del sistema ATM en un MSC-Graph. Las siguientes despliegan los bMSCs que se asocian a los nodos del MSC-Graph.

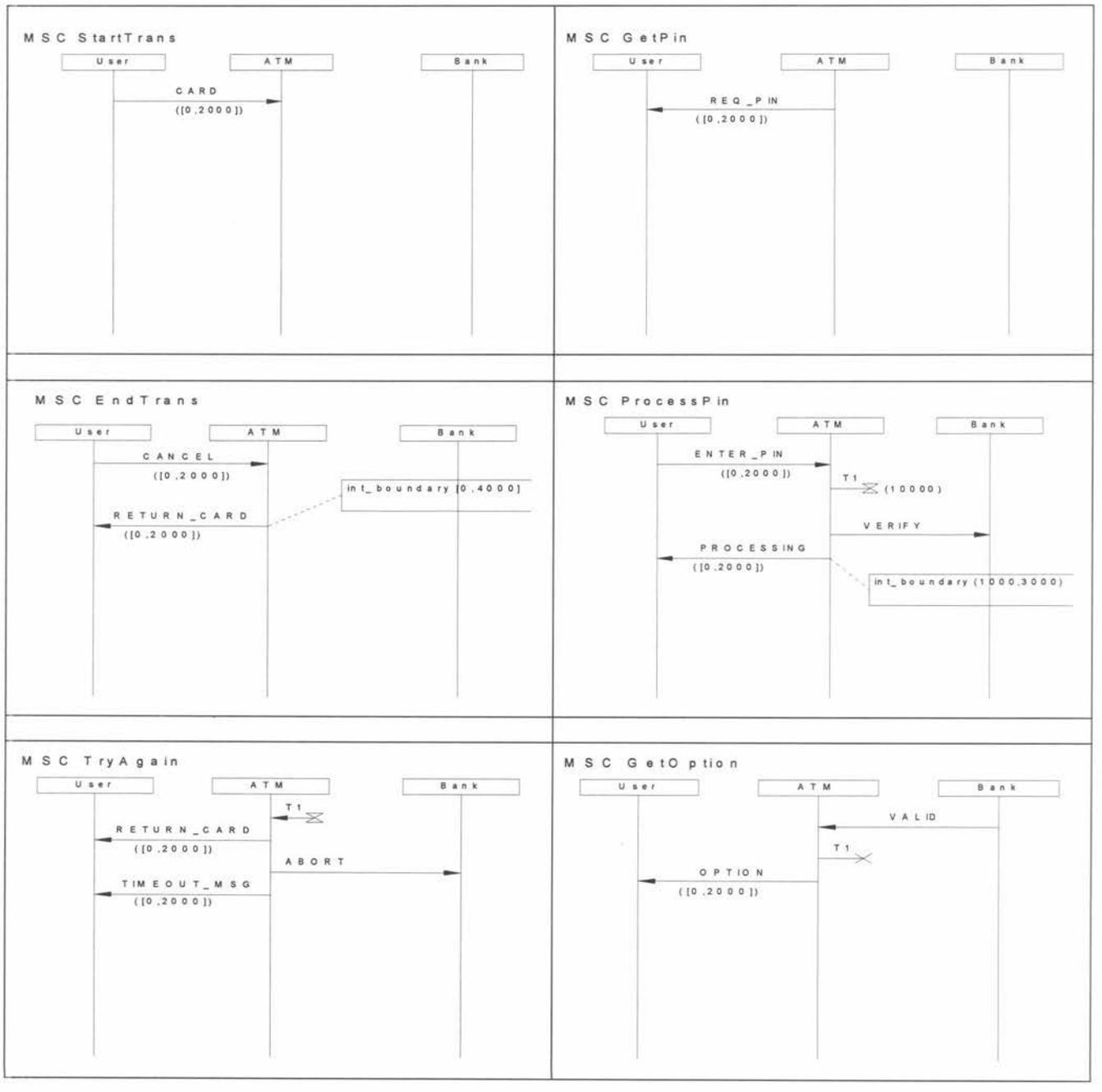


Figura 26

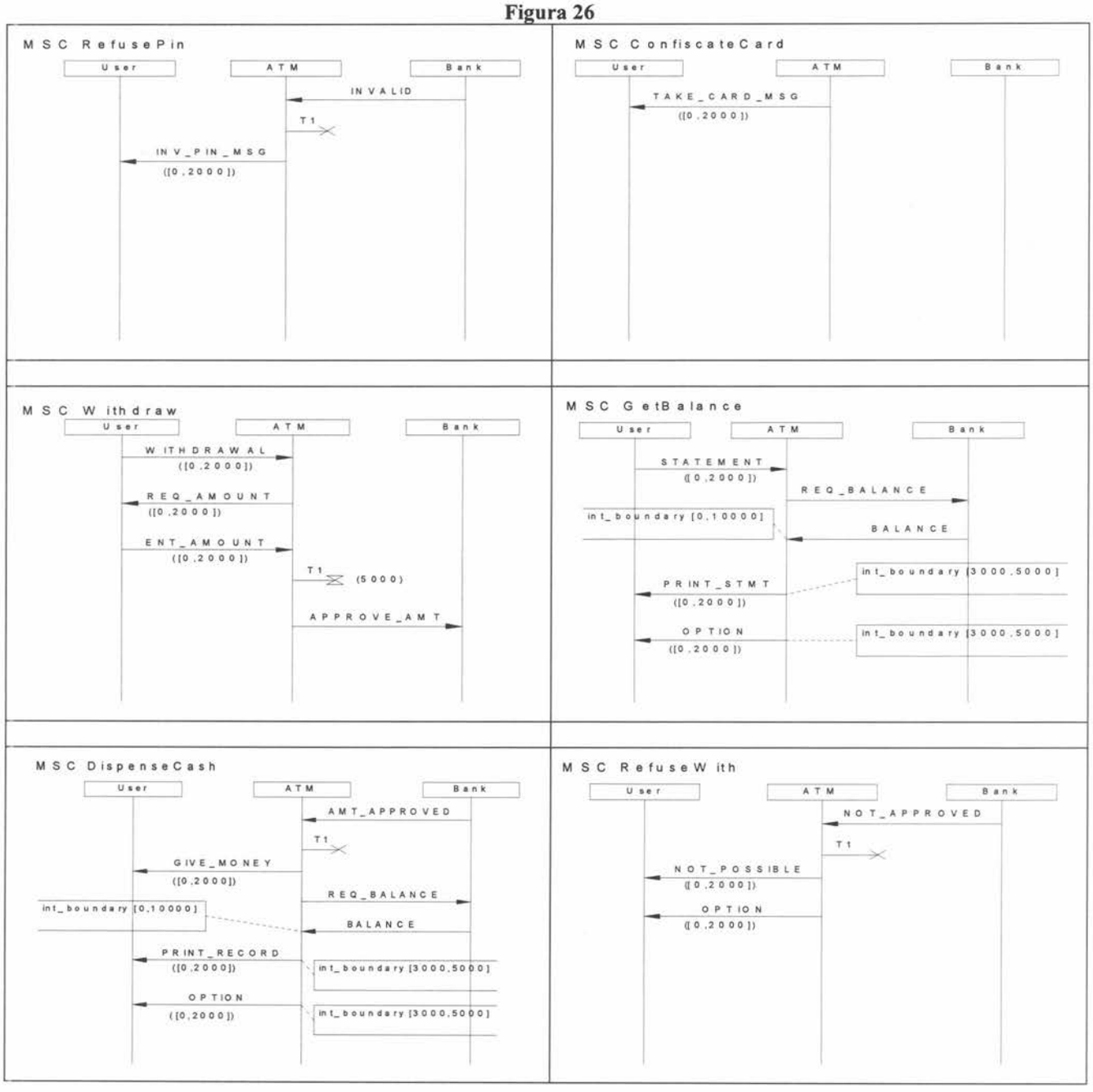


Figura 27

El sistema ATM consiste en tres componentes: un cliente (instancia *User*), el controlador ATM (instancia *ATM*) y el host de un banco (instancia *Bank*).

Inicialmente, el controlador ATM espera a recibir la tarjeta del banco por parte del cliente. Una vez recibida la tarjeta, solicita al cliente el pin correspondiente a la misma. Luego, recibe el pin del cliente para su verificación (bMSC *ProcessPin*); o recibe una solicitud para cancelar la transacción por parte del cliente (bMSC *EndTrans*).

Si el controlador ATM recibe el pin del cliente (bMSC *ProcessPin*), espera hasta 10 segundos para recibir una respuesta desde el banco para validar el mismo. Esta restricción de tiempo se expresa a través del Timer T_1. Si el controlador ATM recibe un pedido para cancelar la transacción (bMSC *EndTrans*), debe devolver la tarjeta del cliente en un intervalo de tiempo de [0, 4] segundos y retornar a su estado inicial.

Cuando T_1 expira por Timeout (bMSC *TryAgain*), la tarjeta del banco del cliente es devuelta, un mensaje apropiado se despliega, y el controlador ATM otra vez retorna a su estado inicial.

La especificación también describe las siguientes restricciones de tiempo: el controlador ATM recibe la aprobación del monto de dinero a retirar en un tiempo restringido por el Timer T_1 con una duración de 5 segundos; el controlador ATM debe enviar un mensaje avisando el procesamiento del pin al cliente entre (1, 3) segundos luego de enviado el pin al banco para su verificación; el controlador ATM espera entre [0, 10] segundos para recibir la respuesta de un pedido de balance desde el banco; el controlador ATM tarda entre [3, 5] segundos en imprimir un recibo, una vez recibida la información de balance desde el banco; el controlador ATM tarda entre [3, 5] segundos en enviar las opciones del menú, una vez impreso el recibo. Además cada comunicación entre el controlador ATM y el cliente (y viceversa) demora [0, 2] segundos.

6.1.1 Informe del resultado obtenido

Un testigo activo

A continuación se muestra el seguimiento de un ejemplo de ejecución, donde el resultado obtenido será un único testigo (CaTSE). Todos los archivos nombrados en este informe, se podrán encontrar junto con el material entregado en el directorio `casos_estudio/ATM`.

A continuación se presenta una tabla que detalla el seguimiento de un caso de ejecución llamado `exec1.log`. La primera columna detalla el número de paso. La segunda columna contiene la entrada del archivo de log. La tercera columna indica la cantidad de testigos activos en el momento de ejecutar el paso del log. Finalmente, la cuarta columna indica qué reacción tiene la herramienta ante la recepción de la entrada en el log.

n	Log	TA	Descripción
1	2002-10-31 17:07:01.200 S(CARD)	1	Se concatena *StartTrans*
2	2002-10-31 17:07:02.200 R(CARD)	1	Es válido el DI MSG [0,2000]
3	2002-10-31 17:07:03.200 S(REQ_PIN)	1	Se concatena *GetPin*
4	2002-10-31 17:07:04.200 R(REQ_PIN)	1	Es válido el DI MSG [0,2000]
5	2002-10-31 17:07:08.200 S(ENTER_PIN)	1	Se concatena *ProcessPin*
6	2002-10-31 17:07:09.200 R(ENTER_PIN)	1	Es válido el DI MSG [0,2000]
7	2002-10-31 17:07:10.200 S(VERIFY)	1	Se encuentra en rango de T1 (10.2 < (09.2+10))
8	2002-10-31 17:07:11.300 S(PROCESSING)	1	Es válido el DI VP (1,3), Se encuentra en rango de T1
9	2002-10-31 17:07:11.700 R(VERIFY)	1	
10	2002-10-31 17:07:12.200 S(VALID)	1	Se concatena *GetOption*
11	2002-10-31 17:07:12.800 R(VALID)	1	Se encuentra en rango de T1
12	2002-10-31 17:07:13.100 S(OPTION)	1	
13	2002-10-31 17:07:13.150 R(PROCESSING)	1	Es válido el DI MSG [0,2000]
14	2002-10-31 17:07:15.099 R(OPTION)	1	Es válido el DI MSG [0,2000]
15	2002-10-31 17:07:18.200 S(WITHDRAWAL)	1	Se concatena *Withdraw*
16	2002-10-31 17:07:19.200 R(WITHDRAWAL)	1	Es válido el DI MSG [0,2000]
17	2002-10-31 17:07:20.200 S(REQ_AMOUNT)	1	
18	2002-10-31 17:07:21.200 R(REQ_AMOUNT)	1	Es válido el DI MSG [0,2000]
19	2002-10-31 17:07:22.200 S(ENT_AMOUNT)	1	
20	2002-10-31 17:07:23.200 R(ENT_AMOUNT)	1	Es válido el DI MSG [0,2000]
21	2002-10-31 17:07:24.200 S(APPROVE_AMT)	1	Se encuentra en rango de T1 (24.2 < (23.2+5))
22	2002-10-31 17:07:25.200 R(APPROVE_AMT)	1	
23	2002-10-31 17:07:26.200 S(NOT_APPROVED)	1	Se concatena *RefuseWith*
24	2002-10-31 17:07:27.200 R(NOT_APPROVED)	1	Se encuentra en rango de T1
25	2002-10-31 17:07:28.200 S(NOT_POSSIBLE)	1	
26	2002-10-31 17:07:29.200 S(OPTION)	1	
27	2002-10-31 17:07:29.200 R(NOT_POSSIBLE)	1	Es válido el DI MSG [0,2000]
28	2002-10-31 17:07:30.200 R(OPTION)	1	Es válido el DI MSG [0,2000]
29	2002-10-31 17:07:31.200 S(CANCEL)	1	Se concatena *EndTrans*
30	2002-10-31 17:07:32.200 R(CANCEL)	1	Es válido el DI MSG [0,2000]
31	2002-10-31 17:07:35.200 S(RETURN_CARD)	1	Es válido el DI VP [0,4000]
32	2002-10-31 17:07:36.200 R(RETURN_CARD)	1	Es válido el DI MSG [0,2000]

n = Número de Paso. TA = Testigos Activos

Al ser alimentada la herramienta con este log, la misma genera tres archivos como resultado.

El primero de ellos es un archivo xml, denominado `exec1_trace.xml`, que representa el Trace del análisis:

```
<ExecutionTrace name="exec1" TimeStamp="2002-11-02 15:20:31.045">
 <ExecutionTime value="0.131 sec." esPrefijo="true"/>
 <WITNESSES MSCGraph="ATM_Example" activos="1">
  <WITNESS MSCGraph="ATM_Example">
   <NodesPath size="8">
    <Node id="NODO_INICIAL" BMSC="NODO_INICIAL"/>
    <Node id="L1" BMSC="StartTrans"/>
    <Node id="L2" BMSC="GetPin"/>
    <Node id="L5" BMSC="ProcessPin"/>
    <Node id="L7" BMSC="GetOption"/>
    <Node id="L11" BMSC="Withdraw"/>
    <Node id="L18" BMSC="RefuseWith"/>
    <Node id="L13" BMSC="EndTrans"/>
   </NodesPath>
   <BMSCMONITOR>
    <BMSC>
    <Instance id="User">
     <EVMSG or="Send" label="CARD" TStamp="2002-10-31 17:07:01.2"/>
     <EVMSG or="Receive" label="REQ_PIN" DI_MSG="[0,2000]" TStamp="2002-10-31 17:07:04.2"/>
     <EVMSG or="Send" label="ENTER_PIN" TStamp="2002-10-31 17:07:08.2"/>
     <EVMSG or="Receive" label="PROCESSING" DI_MSG="[0,2000]" TStamp="2002-10-31 17:07:13.15"/>
     <EVMSG or="Receive" label="OPTION" DI_MSG="[0,2000]" TStamp="2002-10-31 17:07:15.099"/>
     <EVMSG or="Send" label="WITHDRAWAL" TStamp="2002-10-31 17:07:18.2"/>
     <EVMSG or="Receive" label="REQ_AMOUNT" DI_MSG="[0,2000]" TStamp="2002-10-31 17:07:21.2"/>
     <EVMSG or="Send" label="ENT_AMOUNT" TStamp="2002-10-31 17:07:22.2"/>
     <EVMSG or="Receive" label="NOT_POSSIBLE" DI_MSG="[0,2000]" TStamp="2002-10-31 17:07:29.2"/>
     <EVMSG or="Receive" label="OPTION" DI_MSG="[0,2000]" TStamp="2002-10-31 17:07:30.2"/>
     <EVMSG or="Send" label="CANCEL" TStamp="2002-10-31 17:07:31.2"/>
     <EVMSG or="Receive" label="RETURN_CARD" DI_MSG="[0,2000]" TStamp="2002-10-31 17:07:36.2"/>
    </Instance>
    <Instance id="ATM">
     <EVMSG or="Receive" label="CARD" DI_MSG="[0,2000]" TStamp="2002-10-31 17:07:02.2"/>
     <EVMSG or="Send" label="REQ_PIN" TStamp="2002-10-31 17:07:03.2"/>
     <EVMSG or="Receive" label="ENTER_PIN" DI_MSG="[0,2000]" TStamp="2002-10-31 17:07:09.2"/>
     <EVTIM tipo="Set" timer="T1" dur="10000" TStamp="2002-10-31 17:07:09.2"/>
     <EVMSG or="Send" label="VERIFY" TStamp="2002-10-31 17:07:10.2"/>
     <EVMSG or="Send" label="PROCESSING" DI_VP="(1000,3000)" TStamp="2002-10-31 17:07:11.3"/>
     <EVMSG or="Receive" label="VALID" TStamp="2002-10-31 17:07:12.8"/>
     <EVTIM tipo="Reset" timer="T1" TStamp="2002-10-31 17:07:13.1"/>
     <EVMSG or="Send" label="OPTION" TStamp="2002-10-31 17:07:13.1"/>
     <EVMSG or="Receive" label="WITHDRAWAL" DI_MSG="[0,2000]" TStamp="2002-10-31 17:07:19.2"/>
     <EVMSG or="Send" label="REQ_AMOUNT" TStamp="2002-10-31 17:07:20.2"/>
     <EVMSG or="Receive" label="ENT_AMOUNT" DI_MSG="[0,2000]" TStamp="2002-10-31 17:07:23.2"/>
     <EVTIM tipo="Set" timer="T1" dur="5000" TStamp="2002-10-31 17:07:23.2"/>
     <EVMSG or="Send" label="APPROVE_AMT" TStamp="2002-10-31 17:07:24.2"/>
     <EVMSG or="Receive" label="NOT_APPROVED" TStamp="2002-10-31 17:07:27.2"/>
     <EVTIM tipo="Reset" timer="T1" TStamp="2002-10-31 17:07:28.2"/>
     <EVMSG or="Send" label="NOT_POSSIBLE" TStamp="2002-10-31 17:07:28.2"/>
     <EVMSG or="Send" label="OPTION" TStamp="2002-10-31 17:07:29.2"/>
     <EVMSG or="Receive" label="CANCEL" DI_MSG="[0,2000]" TStamp="2002-10-31 17:07:32.2"/>
     <EVMSG or="Send" label="RETURN_CARD" DI_VP="[0,4000]" TStamp="2002-10-31 17:07:35.2"/>
    </Instance>
    <Instance id="Bank">
     <EVMSG or="Receive" label="VERIFY" TStamp="2002-10-31 17:07:11.7"/>
     <EVMSG or="Send" label="VALID" TStamp="2002-10-31 17:07:12.2"/>
     <EVMSG or="Receive" label="APPROVE_AMT" TStamp="2002-10-31 17:07:25.2"/>
     <EVMSG or="Send" label="NOT_APPROVED" TStamp="2002-10-31 17:07:26.2"/>
    </Instance>
   </BMSC>
   <CONFIG>
    <InstanceState name="User" centinel="11" openTimers="{}">
     <EvMsgLastVisited or="Receive" label="RETURN_CARD" DI_MSG="[0,2000]" TStamp="2002-10-31
17:07:36.2"/><EvMsgNextToVisit label="null"/>
    </InstanceState>
    <InstanceState name="ATM" centinel="19" openTimers="{}">
     <EvMsgLastVisited or="Send" label="RETURN_CARD" DI_VP="[0,4000]" TStamp="2002-10-31
17:07:35.2"/>
     <EvMsgNextToVisit label="null"/>
    </InstanceState>
    <InstanceState name="Bank" centinel="3" openTimers="{}">
     <EvMsgLastVisited or="Send" label="NOT_APPROVED" TStamp="2002-10-31 17:07:26.2"/>
     <EvMsgNextToVisit label="null"/>
    </InstanceState>
   </CONFIG>
  </BMSCMONITOR>
 </WITNESS>
</WITNESSES>
</ExecutionTrace>
```

Este archivo de Trace muestra el resultado del análisis del log recibido. En este caso, el resultado ha sido exitoso (se obtuvo un testigo) y el log de ejecución procesado es efectivamente un prefijo válido del lenguaje denotado por el MSC-Graph ATM.

Adicionalmente, si el DebugLevel de la herramienta se establece en 4, el archivo de Trace muestra por cada uno de los testigos que quedaron activos al finalizar el análisis, sus contenidos completos. Este contenido incluye el camino de aceptación recorrido en el MSC-Graph y utilizado para satisfacer el log de ejecución.

El segundo archivo resultante es un archivo de texto, llamado `exec1_bMSCResult1.mpr`, cuyo contenido corresponde al gran bMSC, y que representa en este caso, el único testigo activo. El mismo se graba en formato entendible por Telelogic TAU para ser visualizado con esta herramienta. El archivo resultante se muestra a continuación:

```
mscdocument exec1_bMSCResult1;
msc exec1_bMSCResult1;
ATM: instance;
User: instance;
Bank: instance;
User: out CARD,1([0,2000]) to ATM comment '2002-10-31 17:07:01.2';
ATM: in CARD,1([0,2000]) from User comment '2002-10-31 17:07:02.2';
ATM: out REQ_PIN,2([0,2000]) to User comment '2002-10-31 17:07:03.2';
User: in REQ_PIN,2([0,2000]) from ATM comment '2002-10-31 17:07:04.2';
User: out ENTER_PIN,3([0,2000]) to ATM comment '2002-10-31 17:07:08.2';
ATM: in ENTER_PIN,3([0,2000]) from User comment '2002-10-31 17:07:09.2';
ATM: Set T1/* #SDTMSCPAR(10000)*/;
ATM: out VERIFY,4 to Bank comment '2002-10-31 17:07:10.2';
ATM: out PROCESSING,5([0,2000]) to User comment '2002-10-31 17:07:11.3
int_boundary (1000,3000)';
Bank: in VERIFY,4 from ATM comment '2002-10-31 17:07:11.7';
Bank: out VALID,6 to ATM comment '2002-10-31 17:07:12.2';
ATM: in VALID,6 from Bank comment '2002-10-31 17:07:12.8';
ATM: Reset T1;
ATM: out OPTION,7([0,2000]) to User comment '2002-10-31 17:07:13.1';
User: in PROCESSING,5([0,2000]) from ATM comment '2002-10-31 17:07:13.15';
User: in OPTION,7([0,2000]) from ATM comment '2002-10-31 17:07:15.099';
User: out WITHDRAWAL,8([0,2000]) to ATM comment '2002-10-31 17:07:18.2';
ATM: in WITHDRAWAL,8([0,2000]) from User comment '2002-10-31 17:07:19.2';
ATM: out REQ_AMOUNT,9([0,2000]) to User comment '2002-10-31 17:07:20.2';
User: in REQ_AMOUNT,9([0,2000]) from ATM comment '2002-10-31 17:07:21.2';
User: out ENT_AMOUNT,10([0,2000]) to ATM comment '2002-10-31 17:07:22.2';
ATM: in ENT_AMOUNT,10([0,2000]) from User comment '2002-10-31 17:07:23.2';
ATM: Set T1/* #SDTMSCPAR(5000)*/;
ATM: out APPROVE_AMT,11 to Bank comment '2002-10-31 17:07:24.2';
Bank: in APPROVE_AMT,11 from ATM comment '2002-10-31 17:07:25.2';
Bank: out NOT_APPROVED,12 to ATM comment '2002-10-31 17:07:26.2';
ATM: in NOT_APPROVED,12 from Bank comment '2002-10-31 17:07:27.2';
ATM: Reset T1;
ATM: out NOT_POSSIBLE,13([0,2000]) to User comment '2002-10-31 17:07:28.2';
ATM: out OPTION,14([0,2000]) to User comment '2002-10-31 17:07:29.2';
User: in NOT_POSSIBLE,13([0,2000]) from ATM comment '2002-10-31 17:07:29.2';
User: in OPTION,14([0,2000]) from ATM comment '2002-10-31 17:07:30.2';
User: out CANCEL,15([0,2000]) to ATM comment '2002-10-31 17:07:31.2';
ATM: in CANCEL,15([0,2000]) from User comment '2002-10-31 17:07:32.2';
ATM: out RETURN_CARD,16([0,2000]) to User comment '2002-10-31 17:07:35.2 int_boundary [0,4000]';
User: in RETURN_CARD,16([0,2000]) from ATM comment '2002-10-31 17:07:36.2';
endmsc;
```

A continuación se muestra el archivo `exec1_bMSCResult1.mpr` interpretado por Telelogic TAU:

El nombre del bMSC se compone de la ejecución realizada, agregándole bMSCResultN, donde N representa a cada testigo activo.

El último de los archivos resultantes para el testigo, es el denominado `exec1_MSCGraphResult1.mpr`, que representa el MSCGraph que se genera a partir del bMSC testigo generado.

Si bien este archivo cumple con la sintaxis que utiliza Telelogic TAU, en la versión de la herramienta utilizada, no es posible generar un diagrama a partir de este archivo (sí es posible hacerlo con el bMSC como recién se pudo observar).

El archivo resultante se muestra a continuación:

```
mscdocument exec1_MSCGraphResult1;
msc exec1_MSCGraphResult1;
expr L1;
L1: (StartTrans) seq (L2);
L2: (GetPin) seq (L3);
L3: (ProcessPin) seq (L4);
L4: (GetOption) seq (L5);
L5: (Withdraw) seq (L6);
L6: (RefuseWith) seq (L7);
L7: (EndTrans) seq (L8);
L8: end;
endmsc;
```

Notar que el tiempo total de ejecución del caso fue de 0.131 segundos.
(De la tabla de Trace)

```
  <ExecutionTime value="0.131 sec." esPrefijo="true"/>
```

Adicionalmente, se genera el archivo que representa el cubrimiento sobre el MSC-Graph, con todos aquellos testigos que quedaron activos (aquí es sólo uno).

```
<gxl>
  <node id="L19" bMSC="ProcessPin" covered="false"/>
  <node id="L18" bMSC="RefuseWith" covered="true"/>
  <node id="L17" bMSC="DispenseCash" covered="false"/>
  <node id="L14" bMSC="GetPin" covered="false"/>
  <node id="L13" bMSC="EndTrans" covered="true"/>
  <node id="L23" bMSC="ConfiscateCard" covered="false"/>
  <node id="L9" bMSC="RefusePin" covered="false"/>
  <node id="L12" bMSC="GetBalance" covered="false"/>
  <node id="L8" bMSC="TryAgain" covered="false"/>
  <node id="L11" bMSC="Withdraw" covered="true"/>
  <node id="L22" bMSC="RefusePin" covered="false"/>
  <node id="L7" bMSC="GetOption" covered="true"/>
  <node id="L20" bMSC="EndTrans" covered="false"/>
  <node id="L5" bMSC="ProcessPin" covered="true"/>
  <node id="L4" bMSC="EndTrans" covered="false"/>
  <node id="L2" bMSC="GetPin" covered="true"/>
  <node id="L1" bMSC="StartTrans" covered="true"/>
  <edge begin="L19" end="L7" covered="false"/>
  <edge begin="L19" end="L8" covered="false"/>
  <edge begin="L19" end="L22" covered="false"/>
  <edge begin="L18" end="L11" covered="true"/>
  <edge begin="L18" end="L12" covered="false"/>
  <edge begin="L18" end="L13" covered="true"/>
  <edge begin="L17" end="L11" covered="false"/>
  <edge begin="L17" end="L12" covered="false"/>
  <edge begin="L17" end="L13" covered="false"/>
  <edge begin="L14" end="L19" covered="false"/>
  <edge begin="L14" end="L20" covered="false"/>
  <edge begin="L13" end="L1" covered="true"/>
  <edge begin="L23" end="L1" covered="false"/>
  <edge begin="L9" end="L14" covered="false"/>
  <edge begin="L12" end="L11" covered="false"/>
  <edge begin="L12" end="L12" covered="false"/>
  <edge begin="L12" end="L13" covered="false"/>
  <edge begin="L8" end="L1" covered="false"/>
  <edge begin="L11" end="L17" covered="false"/>
  <edge begin="L11" end="L18" covered="true"/>
  <edge begin="L22" end="L23" covered="false"/>
  <edge begin="L7" end="L11" covered="true"/>
  <edge begin="L7" end="L12" covered="false"/>
  <edge begin="L7" end="L13" covered="true"/>
  <edge begin="L20" end="L1" covered="false"/>
  <edge begin="L5" end="L7" covered="true"/>
  <edge begin="L5" end="L8" covered="false"/>
```

```
  <edge begin="L5" end="L9" covered="false"/>
  <edge begin="L4" end="L1" covered="false"/>
  <edge begin="L2" end="L4" covered="false"/>
  <edge begin="L2" end="L5" covered="true"/>
  <edge begin="L1" end="L2" covered="true"/>
  <edge begin="NODO_INICIAL" end="L1" covered="true"/>
</gxl>
```

6.2 Expendedora de productos (VM)

Como segundo caso de estudio se considera la especificación MSC de una expendedora de productos, de ahora en más llamada VM (Vending Machine), como por ejemplo una máquina expendedora de gaseosas.

Esta especificación fue diseñada para el presente trabajo y el MSC-Graph tiene la particularidad que corresponde a la familia de MSC-Graphs con CIDs. Es de importante interés tratar con especificaciones que posean esta característica porque es uno de los trabajos a futuro propuestos (Ver sección "8 Trabajos a futuro"). A pesar del problema, la herramienta implementada soporta un sublenguaje de aquél que proviene de la especificación, es decir que no lo caracteriza formalmente, pero reconoce las linearizaciones que recorren los CIDs una sóla vez.

La especificación se puede visualizar en las Figura 28 y Figura 29. La primera de ellas representa la especificación del comportamiento del sistema VM en un MSC-Graph. La siguiente despliega los bMSCs que se asocian a los nodos del MSC-Graph.

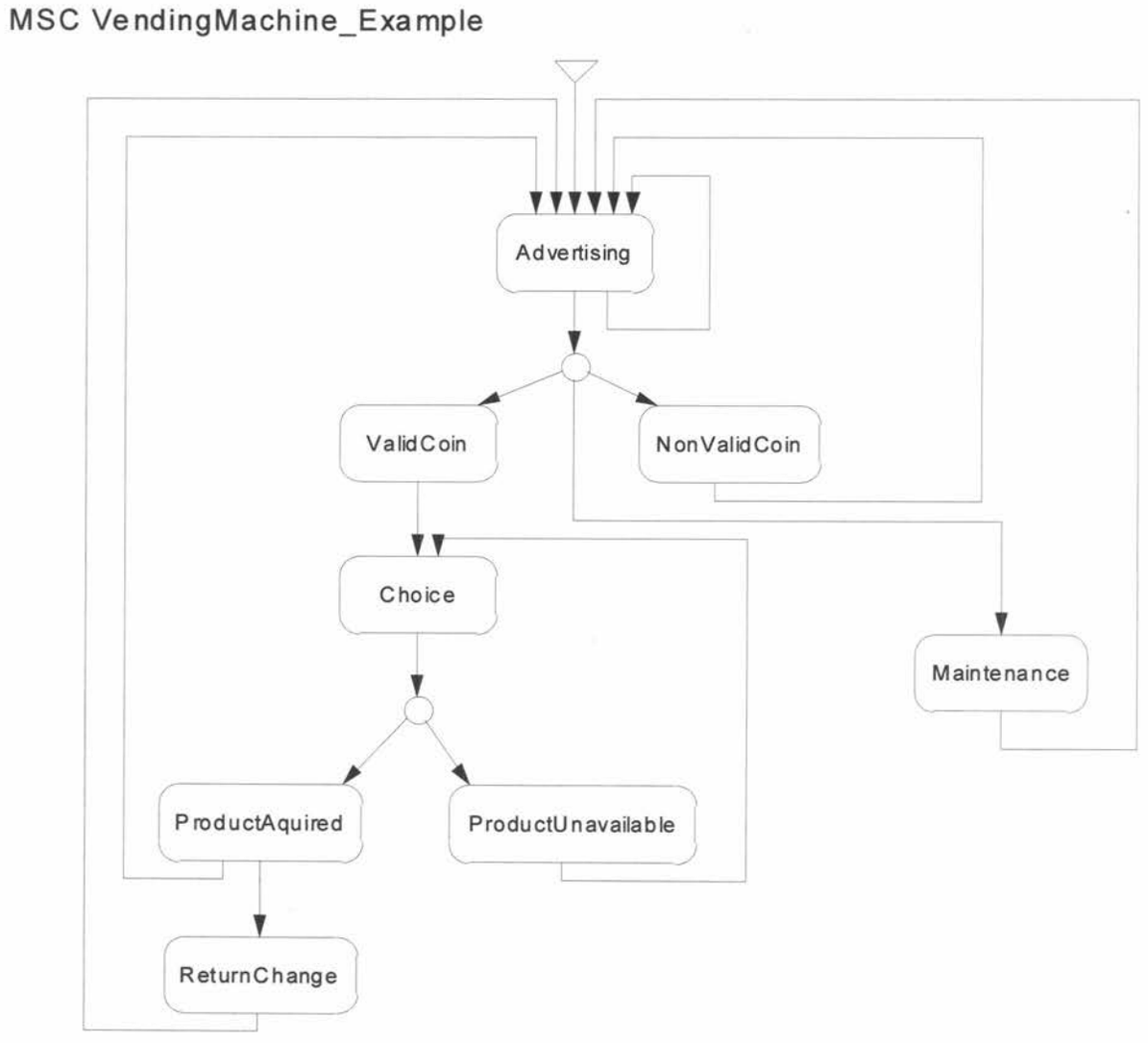


Figura 28

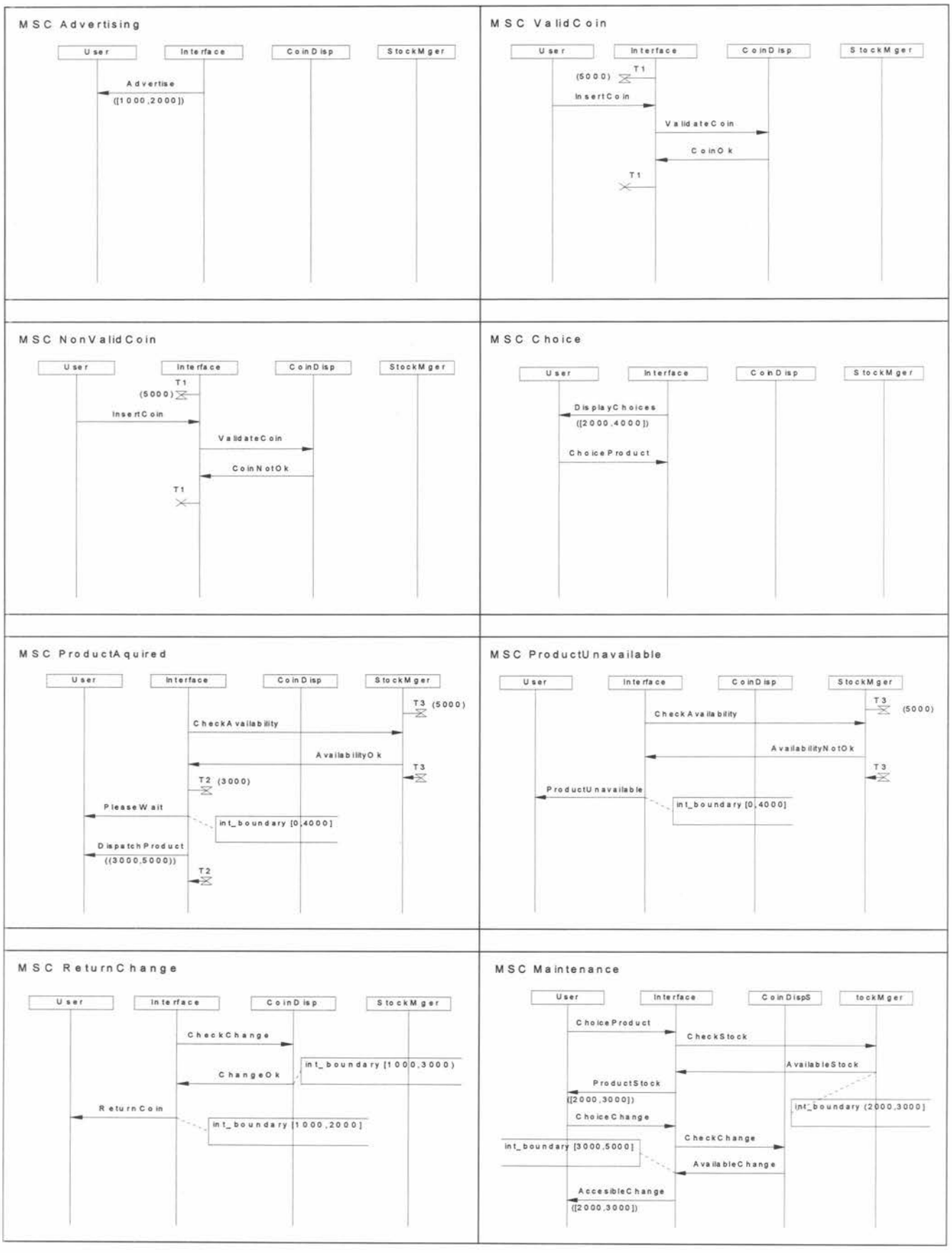

MSC Advertising
User
Interface
CoinDisp
StockMger
Advertise
([1000,2000])
MSC ValidCoin
User
Interface
CoinDisp
StockMger
T1
(5000)
InsertCoin
ValidateCoin
CoinOk
T1
MSC NonValidCoin
User
Interface
CoinDisp
StockMger
T1
(5000)
InsertCoin
ValidateCoin
CoinNotOk
T1
MSC Choice
User
Interface
CoinDisp
StockMger
DisplayChoices
([2000,4000])
ChoiceProduct
MSC ProductAquired
User
Interface
CoinDisp
StockMger
T3 (5000)
CheckAvailability
AvailabilityOk
T3
T2 (3000)
PleaseWait
int_boundary [0,4000]
DispatchProduct
((3000,5000))
T2
MSC ProductUnavailable
User
Interface
CoinDisp
StockMger
T3
(5000)
CheckAvailability
AvailabilityNotOk
T3
ProductUnavailable
int_boundary [0,4000]
MSC ReturnChange
User
Interface
CoinDisp
StockMger
CheckChange
int_boundary [1000,3000)
ChangeOk
ReturnCoin
int_boundary [1000,2000]
MSC Maintenance
User
Interface
CoinDispS
tockMger
ChoiceProduct
CheckStock
AvailableStock
ProductStock
([2000,3000])
int_boundary (2000,3000]
ChoiceChange
CheckChange
int_boundary [3000,5000]
AvailableChange
AccesibleChange
([2000,3000])

Figura 29

El sistema VM consiste en cuatro componentes: un cliente o personal de mantenimiento (instancia *User*), la interface de usuario (instancia *Interface*), el controlador de monedas (instancia *CoinDisp*) y el administrador de stock de productos (instancia *StockMger*).

Mientras la VM no se encuentra en uso, la interface de usuario despliega publicidad hacia el cliente (bMSC *Advertising*). El tiempo de retardo desde que la interface envía una publicidad hasta poder ser visualizada por el cliente, debe cumplir el intervalo de tiempo de [1, 2] segundos.

Luego, la interface de usuario recibe una moneda por parte del cliente o pasa al estado de mantenimiento. Si recibe una moneda, la interface de usuario solicita al controlador de monedas que valide la misma (bMSCs *ValidCoin* y *NonValidCoin*). El tiempo entre que la interface de usuario recibe una moneda y recibe la respuesta de su validación, no debe ser mayor que la duración del Timer T_1 que es de 5 segundos.

Si pasa al estado de mantenimiento (bMSC *Maintenance*), el personal de mantenimiento selecciona algún producto para validar su disponibilidad de stock. La demora del administrador de stock entre que recibe un pedido de chequeo de stock y envía la respuesta debe cumplir con el intervalo de tiempo de [2, 3] segundos. La respuesta enviada de la interface de usuario hacia el personal de mantenimiento no debe demorar más de [2, 3] segundos. También puede validar la disponibilidad de cambio en la VM. La respuesta debe recibirla por parte de la interface de usuario en un intervalo de tiempo entre [3, 5] segundos; y la respuesta enviada hacia el usuario no debe demorar más de [2, 3] segundos.

Si la moneda ingresada por el usuario es válida, el usuario podrá seleccionar un producto (bMSC *Choice*). Para ello, la interface despliega las opciones disponibles en el lapso de tiempo [2, 4] segundos.

El administrador de stock recibe el pedido para chequear la disponibilidad del producto elegido por el usuario y debe responder dentro de la duración del Timer T_3 que es de 5 segundos. Una vez recibida la respuesta del stock del producto seleccionado, la interface de usuario debe en [0, 4] segundos responder al cliente. Si existe disponibilidad, la interface demora hasta la duración del Timer T_2 (3 segundos) para entregar el producto al cliente. Entregado el producto, se chequea la necesidad de entregar un vuelto de dinero, para lo cual el controlador de monedas responde en el intervalo de tiempo de [1, 3] segundos y la interface al usuario en el lapso de [1, 2] segundos.

6.2.1 Informe del resultado obtenido

Dos testigos activos

A continuación se muestra el seguimiento de un ejemplo de ejecución, donde el resultado obtenido serán dos testigos (CaTSEs). Todos los archivos nombrados en este informe, se podrán encontrar junto con el material entregado en el directorio `casos_estudio/VM`.

A continuación se presenta una tabla que detalla el seguimiento de un caso de ejecución llamado `exec5.log`. La primera columna detalla el número de paso. La segunda columna contiene la entrada del archivo de log. La tercera columna indica la cantidad de testigos activos en el momento de ejecutar el paso del log. Finalmente, la cuarta columna indica qué reacción tiene la herramienta ante la recepción de la entrada en el log.

n	Log	TA	Descripción
1	`2002-10-24 07:15:22.000 S(Advertise)`	1	Se concatena *Advertising*
2	`2002-10-24 07:15:23.000 R(Advertise)`	1	Es válido el DI MSG [1000,2000]
3	`2002-10-24 07:15:24.000 S(InsertCoin)`	2	T1: Se concatena *ValidCoin*
			T2: Se concatena *NonValidCoin*
4	`2002-10-24 07:15:25.000 R(InsertCoin)`	2	T1: Se encuentra en rango de T1 (24<(22+5))
			T2: Se encuentra en rango de T1 (24<(22+5))
5	`2002-10-24 07:15:25.500 S(ValidateCoin)`	2	T1: Se encuentra en rango de T1
			T2: Se encuentra en rango de T1
6	`2002-10-24 07:15:26.000 R(ValidateCoin)`	2	T1:
			T2:
7	`2002-10-24 07:15:26.500 S(CoinOk)`	1	T1:
			T2: Se elimina, "La etiqueta actual no fue encontrada"
8	`2002-10-24 07:15:26.800 R(CoinOk)`	1	Se encuentra en rango de T1 (26.8<(22+5))
9	`2002-10-24 07:15:42.000 S(DisplayChoices)`	1	
10	`2002-10-24 07:15:45.000 R(DisplayChoices)`	1	Es válido el DI MSG [2000,4000]
11	`2002-10-24 07:15:46.000 S(ChoiceProduct)`	1	
12	`2002-10-24 07:15:47.000 R(ChoiceProduct)`	1	

13	2002-10-24 07:15:49.000 S(CheckAvailability)	2	T1: Se concatena *ProductAcquired*
			T2: Se concatena *ProductUnavailable*
14	2002-10-24 07:15:50.000 R(CheckAvailability)	2	T1: Se encuentra en rango de T3 (Aqui com.)[7]
			T2: Se encuentra en rango de T3 (Aqui com.)

n = Número de Paso. TA = Testigos Activos

Al ser alimentada la herramienta con este log, la misma genera cinco archivos como resultado.

El primero de ellos es un archivo xml, denominado `exec5_trace.xml`, que representa el Trace del análisis:

```
<ExecutionTrace name="exec5" TimeStamp="2002-11-02 17:01:50.779">
  <Msg value="La etiqueta actual no fue encontrada" tes="NODO_INICIAL,L1,L4" label="CoinOk" or="Send"
InstanceId="CoinDisp" TStamp="2002-10-24 07:15:26.5"/>
  <ExecutionTime value="0.3 sec." esPrefijo="true"/>
  <WITNESSES MSCGraph="VendingMachine_Example" activos="2">
    <WITNESS MSCGraph="VendingMachine_Example">
      <NodesPath size="5">
        <Node id="NODO_INICIAL" BMSC="NODO_INICIAL"/>
        <Node id="L1" BMSC="Advertising"/>
        <Node id="L3" BMSC="ValidCoin"/>
        <Node id="L5" BMSC="Choice"/>
        <Node id="L8" BMSC="ProductAquired"/>
      </NodesPath>
      <BMSCMONITOR>
        <BMSC>
          <Instance id="StockMger">
            <EVTIM tipo="Set" timer="T3" dur="5000" TStamp="2002-10-24 07:15:50.0"/>
            <EVMSG or="Receive" label="CheckAvailability" TStamp="2002-10-24 07:15:50.0"/>
            <EVMSG or="Send" label="AvailabilityOk"/>
            <EVTIM tipo="Timeout" timer="T3"/>
          </Instance>
          <Instance id="User">
            <EVMSG or="Receive" label="Advertise" DI_MSG="[1000,2000]" TStamp="2002-10-24 07:15:23.0"/>
            <EVMSG or="Send" label="InsertCoin" TStamp="2002-10-24 07:15:24.0"/>
            <EVMSG or="Receive" label="DisplayChoices" DI_MSG="[2000,4000]" TStamp="2002-10-24
07:15:45.0"/>
            <EVMSG or="Send" label="ChoiceProduct" TStamp="2002-10-24 07:15:46.0"/>
            <EVMSG or="Receive" label="PleaseWait"/>
            <EVMSG or="Receive" label="DispatchProduct" DI_MSG="(3000,5000)"/>
          </Instance>
          <Instance id="CoinDisp">
            <EVMSG or="Receive" label="ValidateCoin" TStamp="2002-10-24 07:15:26.0"/>
            <EVMSG or="Send" label="CoinOk" TStamp="2002-10-24 07:15:26.5"/>
          </Instance>
          <Instance id="Interface">
            <EVMSG or="Send" label="Advertise" TStamp="2002-10-24 07:15:22.0"/>
            <EVTIM tipo="Set" timer="T1" dur="5000" TStamp="2002-10-24 07:15:22.0"/>
            <EVMSG or="Receive" label="InsertCoin" TStamp="2002-10-24 07:15:25.0"/>
            <EVMSG or="Send" label="ValidateCoin" TStamp="2002-10-24 07:15:25.5"/>
            <EVMSG or="Receive" label="CoinOk" TStamp="2002-10-24 07:15:26.8"/>
            <EVTIM tipo="Reset" timer="T1" TStamp="2002-10-24 07:15:27.0"/>
            <EVMSG or="Send" label="DisplayChoices" TStamp="2002-10-24 07:15:42.0"/>
            <EVMSG or="Receive" label="ChoiceProduct" TStamp="2002-10-24 07:15:47.0"/>
            <EVMSG or="Send" label="CheckAvailability" TStamp="2002-10-24 07:15:49.0"/>
            <EVMSG or="Receive" label="AvailabilityOk"/>
            <EVTIM tipo="Set" timer="T2" dur="3000"/>
            <EVMSG or="Send" label="PleaseWait" DI_VP="[0,4000]"/>
            <EVMSG or="Send" label="DispatchProduct"/>
            <EVTIM tipo="Timeout" timer="T2"/>
          </Instance>
        </BMSC>
        <CONFIG>
          <InstanceState name="StockMger" centinel="1" openTimers="{T3}">
            <EVMsgLastVisited or="Receive" label="CheckAvailability" TStamp="2002-10-24 07:15:50.0"/>
            <EVMsgNextToVisit or="Send" label="AvailabilityOk"/>
          </InstanceState>
          <InstanceState name="User" centinel="3" openTimers="{}">
            <EVMsgLastVisited or="Send" label="ChoiceProduct" TStamp="2002-10-24 07:15:46.0"/>
            <EVMsgNextToVisit or="Receive" label="PleaseWait"/>
          </InstanceState>
          <InstanceState name="CoinDisp" centinel="1" openTimers="{}">
            <EVMsgLastVisited or="Send" label="CoinOk" TStamp="2002-10-24 07:15:26.5"/>
```

[7] El Timestamp del evento Set anterior se asociará al Timestamp actual, ya que no existe evento anterior, por lo que cumplirá perfectamente la restricción establecida por el Timer.

```
        <EVMsgNextToVisit label="null"/>
      </InstanceState>
      <InstanceState name="Interface" centinel="8" openTimers="{}">
        <EVMsgLastVisited or="Send" label="CheckAvailability" TStamp="2002-10-24 07:15:49.0"/>
        <EVMsgNextToVisit or="Receive" label="AvailabilityOk"/>
      </InstanceState>
    </CONFIG>
  </BMSCMONITOR>
</WITNESS>
<WITNESS MSCGraph="VendingMachine_Example">
  <NodesPath size="5">
    <Node id="NODO_INICIAL" BMSC="NODO_INICIAL"/>
    <Node id="L1" BMSC="Advertising"/>
    <Node id="L3" BMSC="ValidCoin"/>
    <Node id="L5" BMSC="Choice"/>
    <Node id="L9" BMSC="ProductUnavailable"/>
  </NodesPath>
  <BMSCMONITOR>
    <BMSC>
      <Instance id="StockMger">
        <EVTIM tipo="Set" timer="T3" dur="5000" TStamp="2002-10-24 07:15:50.0"/>
        <EVMSG or="Receive" label="CheckAvailability" TStamp="2002-10-24 07:15:50.0"/>
        <EVMSG or="Send" label="AvailabilityNotOk"/>
        <EVTIM tipo="Timeout" timer="T3"/>
      </Instance>
      <Instance id="User">
        <EVMSG or="Receive" label="Advertise" DI_MSG="[1000,2000]" TStamp="2002-10-24 07:15:23.0"/>
        <EVMSG or="Send" label="InsertCoin" TStamp="2002-10-24 07:15:24.0"/>
        <EVMSG or="Receive" label="DisplayChoices" DI_MSG="[2000,4000]" TStamp="2002-10-24
07:15:45.0"/>
        <EVMSG or="Send" label="ChoiceProduct" TStamp="2002-10-24 07:15:46.0"/>
        <EVMSG or="Receive" label="ProductUnavailable"/>
      </Instance>
      <Instance id="CoinDisp">
        <EVMSG or="Receive" label="ValidateCoin" TStamp="2002-10-24 07:15:26.0"/>
        <EVMSG or="Send" label="CoinOk" TStamp="2002-10-24 07:15:26.5"/>
      </Instance>
      <Instance id="Interface">
        <EVMSG or="Send" label="Advertise" TStamp="2002-10-24 07:15:22.0"/>
        <EVTIM tipo="Set" timer="T1" dur="5000" TStamp="2002-10-24 07:15:22.0"/>
        <EVMSG or="Receive" label="InsertCoin" TStamp="2002-10-24 07:15:25.0"/>
        <EVMSG or="Send" label="ValidateCoin" TStamp="2002-10-24 07:15:25.5"/>
        <EVMSG or="Receive" label="CoinOk" TStamp="2002-10-24 07:15:26.8"/>
        <EVTIM tipo="Reset" timer="T1" TStamp="2002-10-24 07:15:27.0"/>
        <EVMSG or="Send" label="DisplayChoices" TStamp="2002-10-24 07:15:42.0"/>
        <EVMSG or="Receive" label="ChoiceProduct" TStamp="2002-10-24 07:15:47.0"/>
        <EVMSG or="Send" label="CheckAvailability" TStamp="2002-10-24 07:15:49.0"/>
        <EVMSG or="Receive" label="AvailabilityNotOk"/>
        <EVMSG or="Send" label="ProductUnavailable" DI_VP="[0,4000]"/>
      </Instance>
    </BMSC>
    <CONFIG>
      <InstanceState name="StockMger" centinel="1" openTimers="{T3}">
        <EVMsgLastVisited or="Receive" label="CheckAvailability" TStamp="2002-10-24 07:15:50.0"/>
        <EVMsgNextToVisit or="Send" label="AvailabilityNotOk"/>
      </InstanceState>
      <InstanceState name="User" centinel="3" openTimers="{}">
        <EVMsgLastVisited or="Send" label="ChoiceProduct" TStamp="2002-10-24 07:15:46.0"/>
        <EVMsgNextToVisit or="Receive" label="ProductUnavailable"/>
      </InstanceState>
      <InstanceState name="CoinDisp" centinel="1" openTimers="{}">
        <EVMsgLastVisited or="Send" label="CoinOk" TStamp="2002-10-24 07:15:26.5"/>
        <EVMsgNextToVisit label="null"/>
      </InstanceState>
      <InstanceState name="Interface" centinel="8" openTimers="{}">
        <EVMsgLastVisited or="Send" label="CheckAvailability" TStamp="2002-10-24 07:15:49.0"/>
        <EVMsgNextToVisit or="Receive" label="AvailabilityNotOk"/>
      </InstanceState>
    </CONFIG>
  </BMSCMONITOR>
</WITNESS>
</WITNESSES>
</ExecutionTrace>
```


Este archivo de Trace muestra el resultado del análisis del log recibido. En este caso, el resultado ha sido exitoso porque se encontraron dos testigos. Es decir, que el log procesado es efectivamente un prefijo válido del lenguaje denotado por el MSC-Graph VM y se puede llegar a ese prefijo de dos maneras distintas.

Notar que cuando sucedió el paso 7 (en la segunda línea de `exec5_trace.xml`), se despliega la eliminación del testigo, con un mensaje:

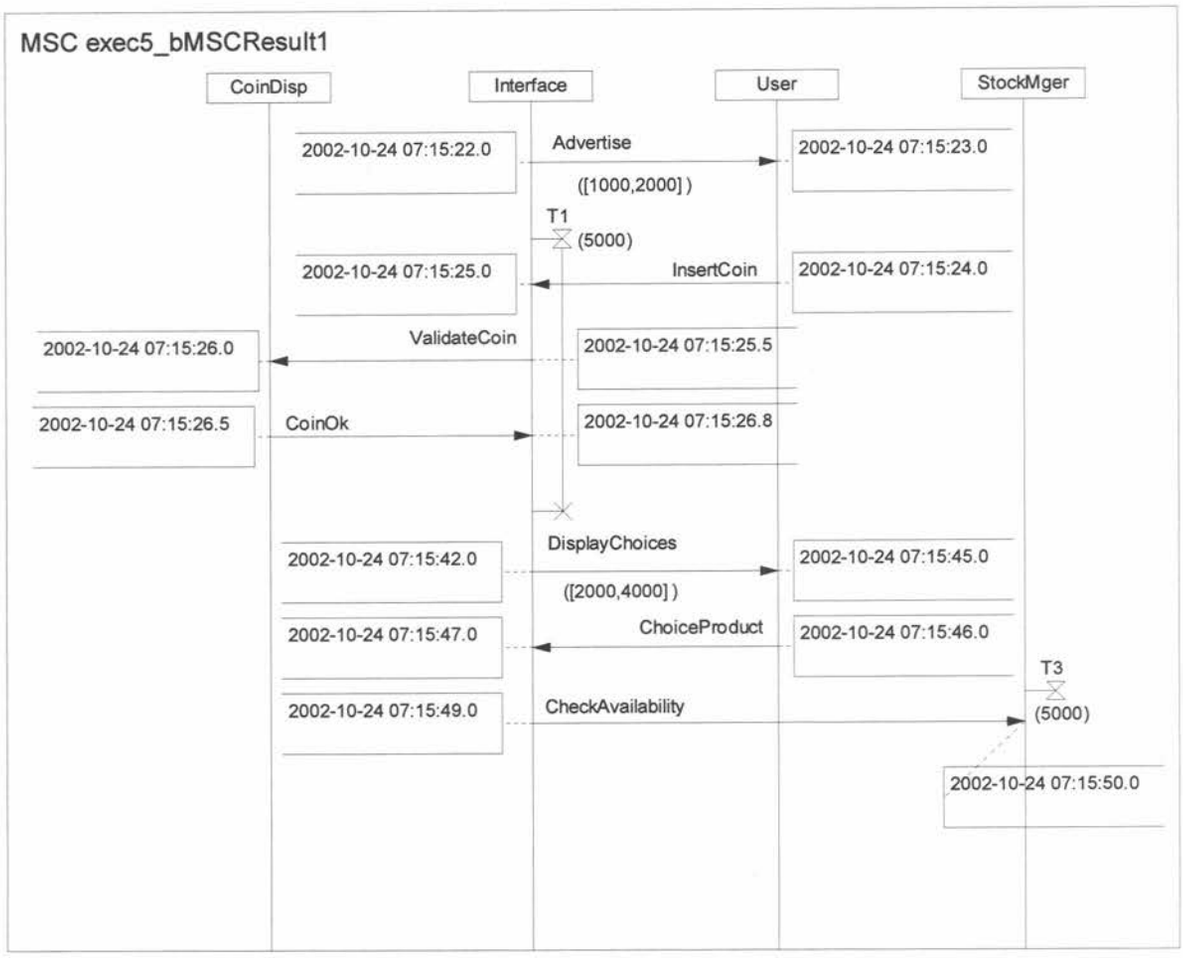
```
<Msg value="La etiqueta actual no fue encontrada" tes="NODO_INICIAL,L1,L4" label="CoinOk" or="Send"
InstanceId="CoinDisp" TStamp="2002-10-24 07:15:26.5"/>
```

Para cada uno de los testigos resultantes, existe un par de archivos. Uno de ellos contiene el gran bMSC, que representa el testigo, y el otro contiene el camino de aceptación en el MSC-Graph que se ha transitado para generarlo.

En este ejemplo, ambos testigos activos tienen el mismo bMSC resultante, ya que los nodos recorridos son similares y los eventos ejecutados en ambos testigos son los mismos, pero se diferencian en el último nodo concatenado en el camino de aceptación en el MSC-Graph. El archivo bMSC generado por la herramienta es el siguiente:

```
mscdocument exec5_bMSCResult1;
msc exec5_bMSCResult1;
CoinDisp: instance;
Interface: instance;
User: instance;
StockMger: instance;
Interface: out Advertise,1([1000,2000]) to User comment '2002-10-24 07:15:22.0';
Interface: Set T1/* #SDTMSCPAR(5000)*/;
User: in Advertise,1([1000,2000]) from Interface comment '2002-10-24 07:15:23.0';
User: out InsertCoin,2 to Interface comment '2002-10-24 07:15:24.0';
Interface: in InsertCoin,2 from User comment '2002-10-24 07:15:25.0';
Interface: out ValidateCoin,3 to CoinDisp comment '2002-10-24 07:15:25.5';
CoinDisp: in ValidateCoin,3 from Interface comment '2002-10-24 07:15:26.0';
CoinDisp: out CoinOk,4 to Interface comment '2002-10-24 07:15:26.5';
Interface: in CoinOk,4 from CoinDisp comment '2002-10-24 07:15:26.8';
Interface: Reset T1;
Interface: out DisplayChoices,5([2000,4000]) to User comment '2002-10-24 07:15:42.0';
User: in DisplayChoices,5([2000,4000]) from Interface comment '2002-10-24 07:15:45.0';
User: out ChoiceProduct,6 to Interface comment '2002-10-24 07:15:46.0';
Interface: in ChoiceProduct,6 from User comment '2002-10-24 07:15:47.0';
Interface: out CheckAvailability,7 to StockMger comment '2002-10-24 07:15:49.0';
StockMger: Set T3/* #SDTMSCPAR(5000)*/;
StockMger: in CheckAvailability,7 from Interface comment '2002-10-24 07:15:50.0';
endmsc;
```

A continuación se muestra el bMSC resultante visualizado a través de Telelogic TAU:



Para el Testigo resultante 1, el camino del grafo es:

```
mscdocument exec5_MSCGraphResult1;
msc exec5_MSCGraphResult1;
expr L1;
L1: (Advertising) seq (L2);
L2: (ValidCoin) seq (L3);
L3: (Choice) seq (L4);
L4: (ProductAquired) seq (L5);
L5: end;
endmsc;
```

mientras que para el Testigo resultante 2, el camino es:

```
mscdocument exec5_MSCGraphResult2;
msc exec5_MSCGraphResult2;
expr L1;
L1: (Advertising) seq (L2);
L2: (ValidCoin) seq (L3);
L3: (Choice) seq (L4);
L4: (ProductUnavailable) seq (L5);
L5: end;
endmsc;
```

Nótese la diferencia en el nodo L6, siendo en (1) el bMSC *ProductAcquired*, mientas que en (2) es el bMSC *ProductUnavailable*.

Notar que el tiempo total de ejecución del caso fue de 0.3 segundos.
(De la tabla de Trace)

```
<ExecutionTime value="0.3 sec." esPrefijo="true"/>
```

Sin testigos activos

A continuación se muestra el seguimiento de un ejemplo de ejecución, donde como resultado no se obtendrán testigos (CaTSEs). Todos los archivos nombrados en este informe, se podrán encontrar junto con el material entregado en el directorio `casos_estudio/VM`.

El log de ejecución utilizado para el seguimiento, es el archivo `exec6.log` (es similar a `exec5.log`, pero agregando una nueva etiqueta). Su contenido se puede visualizar en la segunda columna de la siguiente tabla, que describe el desarrollo de la ejecución:

n	Log	TA	Desc
1	2002-10-24 07:15:22.000 S(Advertise)	1	Se concatena *Advertising*
2	2002-10-24 07:15:23.000 R(Advertise)	1	Es válido el DI MSG [1000,2000]
3	2002-10-24 07:15:24.000 S(InsertCoin)	2	T1: Se concatena *ValidCoin*
			T2: Se concatena *NonValidCoin*
4	2002-10-24 07:15:25.000 R(InsertCoin)	2	T1: Se encuentra en rango de T1 (24<(22+5))
			T2: Se encuentra en rango de T1 (24<(22+5))
5	2002-10-24 07:15:25.500 S(ValidateCoin)	2	T1: Se encuentra en rango de T1
			T2: Se encuentra en rango de T1
6	2002-10-24 07:15:26.000 R(ValidateCoin)	2	
7	2002-10-24 07:15:26.500 S(CoinOk)	1	
			T2: Se elimina, "La etiqueta actual no fue encontrada"
8	2002-10-24 07:15:26.800 R(CoinOk)	1	Se encuentra en rango de T1 (26.8<(22+5))
9	2002-10-24 07:15:42.000 S(DisplayChoices)	1	
10	2002-10-24 07:15:45.000 R(DisplayChoices)	1	Es válido el DI MSG [2000,4000]
11	2002-10-24 07:15:46.000 S(ChoiceProduct)	1	
12	2002-10-24 07:15:47.000 R(ChoiceProduct)	1	
13	2002-10-24 07:15:49.000 S(CheckAvailability)	2	T1: Se concatena *ProductAcquired*
			T2: Se concatena *ProductUnavailable*
14	2002-10-24 07:15:50.000 R(CheckAvailability)	2	T1: Se encuentra en rango de T3 (Aqui com.)
			T2: Se encuentra en rango de T3 (Aqui com.)
15	2002-10-24 07:15:56 S(AvailabilityOk)	0	T1: Se elimina, "Timestamp de etiqueta actual no en rango de un Timer activo".
			T2: Se elimina, "La etiqueta actual no fue encontrada".

n = Número de Paso. TA = Testigos Activos

Al ser alimentada la herramienta con este log, la misma genera un archivo como resultado denominado `exec6_trace.xml`, que representa el Trace del análisis:

```
<ExecutionTrace name="exec6" TimeStamp="2002-11-02 17:38:44.718">
<Msg value="La etiqueta actual no fue encontrada" tes="NODO_INICIAL,L1,L4" label="CoinOk" or="Send"
InstanceId="CoinDisp" TStamp="2002-10-24 07:15:26.5"/>
<Msg value="TSTamp de etiqueta actual no en rango de un Timer activo" tes="NODO_INICIAL,L1,L3,L5,L8"
label="AvailabilityOk" or="Send" InstanceId="StockMger" TStamp="2002-10-24 07:15:56.0"/>
<Msg value="La etiqueta actual no fue encontrada" tes="NODO_INICIAL,L1,L3,L5,L9"
label="AvailabilityOk" or="Send" InstanceId="StockMger" TStamp="2002-10-24 07:15:56.0"/>
<ExecutionTime value="0.04 sec." esPrefijo="false"/>
<WITNESSES MSCGraph="VendingMachine_Example" activos="0">
</WITNESSES>
</ExecutionTrace>
```

Este archivo de Trace muestra el resultado del análisis del log recibido. En este caso, no existe ningún posible testigo (CaTSE) para el log, es decir, que la secuencia de etiquetas recibida no es un prefijo válido del lenguaje denotado por el MSC-Graph VM.

El motivo por el que falló cada uno de los testigos que en algún momento estuvieron activos, aparece detallado en el archivo de trace.

Notar que el tiempo total de ejecución del caso fue de 0.04 segundos.
(De la tabla de Trace)

```
<ExecutionTime value="0.04 sec." esPrefijo="false"/>
```

Adicionalmente, se genera el archivo que representa el cubrimiento sobre el MSC-Graph, con todos aquellos testigos que quedaron activos obtenidos de la ejecución de "Dos testigos activos" (exec5.log) y "Sin testigos activos" (exec6.log).


```
<gxl>
  <node id="L9" bMSC="ProductUnavailable" covered="true"/>
  <node id="L8" bMSC="ProductAquired" covered="true"/>
  <node id="L10" bMSC="ReturnChange" covered="false"/>
  <node id="L6" bMSC="Maintenance" covered="false"/>
  <node id="L5" bMSC="Choice" covered="true"/>
  <node id="L4" bMSC="NonValidCoin" covered="false"/>
  <node id="L3" bMSC="ValidCoin" covered="true"/>
  <node id="L1" bMSC="Advertising" covered="true"/>
  <edge begin="L9" end="L5" covered="true"/>
  <edge begin="L8" end="L1" covered="true"/>
  <edge begin="L8" end="L10" covered="false"/>
  <edge begin="L10" end="L1" covered="false"/>
  <edge begin="L6" end="L1" covered="false"/>
  <edge begin="L5" end="L8" covered="true"/>
  <edge begin="L5" end="L9" covered="true"/>
  <edge begin="L4" end="L1" covered="false"/>
  <edge begin="L3" end="L5" covered="true"/>
  <edge begin="L1" end="L1" covered="true"/>
  <edge begin="L1" end="L3" covered="true"/>
  <edge begin="L1" end="L4" covered="false"/>
  <edge begin="L1" end="L6" covered="false"/>
  <edge begin="NODO_INICIAL" end="L1" covered="true"/>
</gxl>
```

7 Conclusiones y aportes

Utilizamos MSCs, un formalismo muy conocido en las metodologías de especificación, diseño, simulación y documentación de sistemas.

Los sistemas comerciales actuales que se utilizan para diseñar MSCs no poseen como funcionalidad el chequeo de comportamiento observado del software contra el especificado en MSCs.

Definimos formalmente la sintaxis y semántica de bMSCs y MSC-Graphs basados en la literatura, con el agregado formal de requerimientos de tiempo (Delay Intervals y Timers). La semántica de bMSCs y MSC-Graphs se definió en términos de un lenguaje formal temporizado (secuencias aceptables de eventos con timestamps).

Desarrollamos a nivel teórico una herramienta para monitorear sistemas utilizando especificaciones MSC-Graphs. Definimos algoritmos tales que, dados una especificación MSC-Graph y un log de ejecución, detectan si el log responde a un comportamiento aceptable de la especificación. Se permite el chequeo de comportamientos de modelos de software contra el comportamiento especificado en MSCs, detectando comportamientos inesperados o no especificados, así como violaciones a los requerimientos de tiempo establecidos.

Un trabajo relacionado con nuestro desarrollo puede verse en [AY99] que presenta el análisis de fomalizar el problema de *model checking* utilizando teoría de autómatas para verificaciones formales, donde el sistema verificado es descripto por un bMSC, un MSC-Graph o un HMSC. En el trabajo se describen formalmente bMSCs, MSC-Graphs y HMSCs (MSC jerárquico cuyos nodos son bMSCs u otro MSC jerárquico). Para los MSC-Graphs se describen formalmente concatenaciones sincrónicas y asincrónicas. Junto con las descripciones, se presenta un *model checking* para bMSCs, otro para MSC-Graphs y otro para HMSCs. Un requerimiento natural del análisis del paper, es la propiedad de boundedness que deben tener los MSC-Graphs, cuyas concatenaciones de bMSCs se interpreten en forma asincrónica, para verificar que el lenguaje a construir es regular y puede ser reconocido por un autómata. La ejecución que realiza el modelo es completa (no acepta validar prefijos) y **no se utilizan definiciones de tiempo**.

En cambio en el presente trabajo, no es problema la condición de boundedness de los MSC-Graphs para poder realizar el análisis propuesto. A su vez, la ejecución es parcial (un prefijo se considera válido) y on the fly, y se permiten definiciones de tiempo para establecer restricciones de tiempo.

Generamos reportes sobre el cubrimiento de los nodos y ejes del MSC-Graph recibido, de acuerdo al análisis de los logs de ejecución procesados. Esta funcionalidad permite conocer aquellas secciones de la especificación que no fueron ejecutadas.

Implementamos la herramienta desarrollada a nivel teórico, utilizando una arquitectura y diseño que permiten crecer en diferentes aspectos, como ser la implementación de trabajos a futuro relacionados. La herramienta utiliza un sublenguaje de la gramática estándar definida por Z.120, para generar las especificaciones y los resultados obtenidos del monitoreo. Detallamos la arquitectura de la herramienta a nivel conceptual y de implementación, así como el diseño y estructuras de representación utilizados.

Hemos presentado casos de estudio describiendo datos relevantes de los mismos, y los resultados al ser ejecutados en la herramienta.

Finalmente, consideramos que realizar monitoreo de sistemas temporizados especificados con MSC-Graphs genera un aporte valorable al momento de la detección de errores a través de formalismos visuales. Especialmente por utilizarse una herramienta difundida en el campo de desarrollo de software y las telecomunicaciones, donde su principal ventaja es la clara disposición gráfica en su representación visual, permitiendo una comprensión intuitiva del comportamiento del sistema descripto.

8 Trabajos a futuro

A continuación se decriben distintos trabajos que se consideran pueden realizarse a futuro.

1. Implementar CIDs

Implementar la herramienta para que el monitoreo de MSC-Graphs soporte ciclos con condiciones especiales como instancias Idle. La funcionalidad que se debe modificar, es la operación buscarExtendido, que debe solucionar el problema de encontrar los posibles CHE (Caminos hacia una Etiqueta) teniendo en cuenta los bMSCs de los nodos que pertenecen a algún CID de un MSC-Graph. Esta mejora modifica el comportamiento de la herramienta.

Definición 24: CaTSE que utiliza CIDs

Sea MSC-Graph $G = (B, V^I, N, suc, getbMSC)$ que posee algún CID $C = v_i \dots v_k$ con $0 \leq i, k \leq \#V$.
Sea T CaTSE σ en G y $Nr = v_0 \dots v_p$ con $0 \leq p \leq \#V$ los nodos recorridos de T en G.
Si T no es CaTSE de $(\sigma \bullet a)$ en G y $Ult(C) = v_k \in Nr$.
Entonces se define $Nr = Nr_1$ & Nr_2 donde $Nr_1 = v_0 \dots v_k$ y $Nr_2 = v_s \dots v_p$ (v_k es la última ocurrencia del nodo y v_s el siguiente nodo a esa última ocurrencia de v_k en Nr). Nr_1 está formado por los nodos recorridos de T hasta la última ocurrencia del nodo v_k y Nr_2 con el resto de los nodos, tal que Nr_1 y Nr_2 concatenados forman Nr.
Se observa que el nuevo T' CaTSE formado por los nodos recorridos $Nr' = Nr_1$ & C & Nr_2, se debe tener en cuenta para validar si es un posible candidato CaTSE $(\sigma \bullet a)$.

El problema de dar infinitas vueltas en un CID, se resolvería devolviendo un ***Camino canónico***, es decir un camino que representará a todos aquellos que posean el CID una o muchas veces. En el primer caso, el ***Camino canónico*** será el camino con menor cantidad de CIDs, es decir el que pasa una sóla vez por él. En el caso que el ***Camino canónico*** deje de ser un CaTSE válido, se podrá generar un posible nuevo CaTSE como lo describe la Definición 24 y será el nuevo ***Camino canónico*** que representará también a aquellos que poseen por lo menos una vuelta más que él.

Por ejemplo en la Figura 14, los Caminos de la forma:

M1, *M2*, *M3*, M4
M1, *M2*, *M3*, *M1*, *M2*, *M3*, M4
M1, *M2*, *M3*, *M1*, *M2*, *M3*, *M1*, *M2*, *M3*, ..., *M1*, *M2*, *M3*, M4

Serán representados por el ***Camino canónico*** *M1*, *M2*, *M3*, M4.

Esta modificación agrega una alternativa más en el momento de buscar un CHE. Hasta el momento existían dos operaciones: el buscar local dentro de un bMSC y el buscarExtendido. A partir de ahora se debe tener en cuenta también, la posible extensión de un CaTSE, utilizando todos los CIDs que se encuentren en el MSC-Graph. Es decir que el CaTSE podrá extenderse no sólo hacia el final del mismo, sino que también podrá hacerlo en aquellos lugares que representan un CID en el MSC-Graph.

Observación: al buscar posibles CHEs utilizando CIDs y al extender la configuración de un CaTSE con ellos, ésta debe continuar siendo una configuración. Para ello, la concatenación intermedia de los bMSCs en el CaTSE, sólo podrá realizarse en el caso de que no hayan ocurrido luego de la concatenación, eventos posteriores en las instancias que no son Idle en el CID.

2. Mejoras en las estructuras

Optimizar las estructuras utilizando otras que permitan **compartir datos** y mejorar el espacio de almacenamiento. Por ejemplo, hoy se utiliza una representación trivial (secuencia) para los testigosAct del tipo *Testigos*, y podría modificarse para almacenarlos en un árbol, de manera que toda sección común sea una única rama del árbol.

Si se desea que la herramienta no genere un trace del seguimiento realizado, como así también del bMSC correspondiente a la ejecución de la secuencia analizada, se podría realizar una **foresta de testigos**, es decir eliminar toda sección validada en forma completa que no aporta información seguir manteniéndola.

Una implementación para la foresta de testigos consiste en eliminar la porción de los bMSCs de CaTSEs, tal que toda restricción de tiempo presente en él se haya validado en forma completa y sus eventos hayan sido marcados.

Una restricción de tiempo correspondiente a un Delay Interval se dice que fue validada en forma completa, si y sólo si se validó la misma contra el Timestamp asociado al segundo evento del Delay Interval. Es decir que ambos eventos asociados al Delay Interval ocurrieron y la restricción de tiempo fue validada.

Una restricción de tiempo correspondiente a un Timer se dice que fue validada en forma completa, si y sólo si se validó la misma contra todos los eventos ocurridos durante la duración del Timer y contra el próximo evento inmediatamente posterior al Timeout del mismo.

Se pide que toda restricción de tiempo haya sido validada, por el simple hecho que al ser eliminada no podrá ser tenida en cuenta nuevamente.

Tanto la mejora de las estructuras como la foresta de testigos es transparente para el comportamiento de la herramienta y no afecta el algoritmo principal propuesto. Sólo afecta a las estructuras elegidas para la implementación de un CaTSE, permitiendo mejorar el espacio de almacenamiento.

3. Timers Flotantes

Modificar el seteo actual de Timestamps de eventos de Timer, que los asocia a eventos visibles de mensaje inmediatamente anterior, a **Timestamps flotantes**. Es decir que los eventos de Timer podrán suceder entre el Timestamp del evento de mensaje inmediatamente anterior y posterior al mismo.

Por lo tanto, el análisis y la implementación de esta funcionalidad implicará la resolución de un sistema lineal para cada restricción de tiempo definida por un Timer y modificará el comportamiento de la herramienta.

4. Restricciones de tiempo globales

Mejorar el **análisis inicial**, para descubrir con anticipación un mal comportamiento del sistema analizado, adelantándose a futuras violaciones en las restricciones de tiempo. Es decir, se podría analizar en todo momento, dados los Timestamps de los eventos ocurridos y las restricciones de tiempo que se deberán validar, si las mismas podrán ser satisfechas o no. En el caso de que no sea posible, el sistema se encuentra hasta el momento en un comportamiento válido del sistema, pero no será válido teniendo en cuenta la totalidad de la especificación.

5. Caché de CHEs

Mejorar la implementación actual, almacenando las búsquedas extendidas realizadas (para encontrar los CHEs cuando las mismas no se encuentran localmente en el bMSC monitoreado del testigo) en una estructura que permita un fácil acceso a las mismas, si son requeridas nuevamente. De esta forma, se logra tener un **caché** con búsquedas que si deben realizarse en forma repetitiva, consumen un tiempo considerable de cómputo.

6. Multithreading

Multithreading en el análisis de potenciales testigos de la secuencia de etiquetas. Es decir que dada una nueva etiqueta, se podría lanzar en forma paralela la búsqueda de los nuevos testigos a partir de los testigos activos. Esto mejoraría al ciclo interno del algoritmo presentado.

9 Agradecimientos

Agradecemos por tanto apoyo incondicional y paciencia a nuestros amores Adriana y Yanina.

También por el apoyo en nuestra educación a Juan Carlos, Alcira, Paola, Armida, Beatriz y familiares en general, porque a ellos les robamos el tiempo incurrido en la investigación y desarrollo de nuestro trabajo.

A muchos compañeros de trabajo que han aportado consejos y soluciones a algunos problemas. Destacamos especialmente a Bauna, Aurelio, Darío y Silvina.

A Víctor Braberman por su seriedad, seguimiento, dedicación y profesionalidad en la dirección de esta tesis. Su trabajo excedió nuestras expectativas como director del presente trabajo.

Y a todos aquellos que también confiaron en nosotros. ☺

10 Referencias

[AEY00] R. Alur, K. Etessami and M. Yannakakis, **Inference of Message Sequence Charts**, Bell Labs, September 2000. Proceedings of the 22nd International Conference on Software Engineering (ICSE'00).

[AHP96] R. Alur, G. Holzmann and D. Peled, **An Analyzer for Message Sequence Charts**, in Software Concepts amd Tools, 1996.

[AY99] R. Alur and M. Yannakakis., **Model checking of message sequence charts**, Proceedings of the Tenth International Conference on Concurrency Theory, LNCS 1664, Springer Verlag, 1999.

[BB87] T. Bolognesi and E. Brinksma, **Introduction to the ISO specification language LOTOS**. Computer Networks, 1987.

[BJR97] G. Booch, I. Jacobson and J. Rumbaugh, **Unified Modeling Language User Guide**. Addison Wesley, 1997.

[BL97] H. Ben-Abdallah and S. Leue, **Timing Constraints in Message Sequence Chart Specifications**, University of Waterloo, Waterloo, 1997. Proceedings of the Tenth International Conference on Formal Description Techniques FORTE/PSTV'97, Osaka, Japan, 1997.

[GMP01] E. Gunter, A. Muscholl and D. Peled, **Compositional Message Sequence Charts**, TACAS 2001.

[HW00] R. Holt and A. Winter, **A Short Introduction to the GXL Software Exchange Format**, 2000.

[JAVCC] MetaMata, **Java Compiler Compiler 2.1**, http://www.metamata.com/javacc/

[LL99] X. Li and J. Lilius, **Timing Analysis of Message Sequence Charts**, TUCS Technical Report No. 255, Turku Centre for Computer Science, Turku, Finland, 1999.

[MP00] A. Muscholl and D. Peled, **Analyzing Message Sequence Charts**, LIAFA, Université Paris 7, 2000.

[MR97] S. Mauw and M.A. Reniers, **High-Level Message Sequence Charts**, Eindhoven Univ. Of Tech., The Netherlands, 1997. In Proceedings of SDL '97: Time for Testing {SDL, MSC and Trends}. Elsevier, 1997.

[MSC92] ITU-T, **Z.120 Message Sequence Charts** (MSC'92), 1992.

[MSC96] ITU-T, **Z.120 Message Sequence Charts** (MSC'96), 1996.

[MSC98] ITU-T, **Z.120 Anexo B** : Formal Semantics of Message Sequence Charts . ITU-T, 1998.

[MSC99] ITU-T, **Z.120 Message Sequence Charts** (MSC'99), 1999.

[RGG96] E. Rudolph, P. Graubmann, J. Grabowski, **Tutorial on Message Sequence Charts**, Computer Networks and ISDN Systems, 1996.

[RHAP] I-Logix, **Rhapsody,** http://www.ilogix.com/products/rhapsody/rhap_inj.cfm

[RT] Rational, **Rational Rose**, http://www.rational.com/products/rose/real_time/rtrose.jsp

[RTDS] Pragmadev, **Real Time Development Studio**, http://www.pragmadev.com/support/downloads.html

[SDL94] ITU-T, **CCITT Specification and Description Language (SDL)**, 1994.

[SGW94] B. Selic, G. Gullekson and P. Ward, **Real-Time object oriented modeling and design**. J. Wiley, 1994.

[TAU] Copyright © 2001 Telelogic AB, **Telelogic Tau SDL Suite 4.2**, http://www.telelogic.com/products/tau/

[UMK01] S. Uchitel, J. Magee and J. Kramer, **Detecting Implied Scenarios in MSCs Using LTSA.**, Department of Computing, Imperial College, London, 2001.

[Z95] J. Woodcock and J. Davies, **Using Z - Specification, Refinement, and Proof**, University of Oxford, 1995.

11 Anexo A

11.1 Operaciones Auxiliares

11.1.1 Prefijos y Prefijar

Dado e $\in \Sigma$:

- **Prefijos: seq(Σ) $\rightarrow$ conj[seq(Σ)]**
  Prefijos([]) = {[]}
  Prefijos(e.S) = {e.S} $\cup$ Prefijos(S)

- **Prefijar: conj[seq(Σ)] $\rightarrow$ conj[seq(Σ)]**
  Prefijar({}) = {}
  Prefijar(C) = $\cup_{S \in C}$ (Prefijos(S))

11.2 Definición de la Sintaxis Concreta

11.2.1 Introducción al metalenguaje de la gramática textual

En la forma Backus-Naur (BNF, Backus-Naur form) un símbolo terminal se indica sin ser encerrado entre paréntesis angulares (< y >).

Los paréntesis angulares y sus palabras interiores representan un símbolo no terminal. Para cada símbolo no terminal se define una regla de producción en forma de gramática textual concreta. Por ejemplo:

```
<instance parameter decl> ::= inst <instance parm decl list> <end>
```

Una regla de producción para un símbolo no terminal se compone del símbolo no terminal situado a la izquierda del símbolo ::=, y de uno o más constructivos constituidos por uno o más símbolos no terminales y/o terminales colocados a la derecha. Por ejemplo, <instance parameter decl> e <instance parm decl list> y <end> en el ejemplo anterior son símbolos no terminales; **inst** es un símbolo terminal.

A veces un símbolo incluye una parte subrayada. Esta parte subrayada destaca un aspecto semántico de ese símbolo, por ejemplo <msc name> es sintácticamente idéntico a <name> pero semánticamente requiere que el nombre sea un nombre de bMSC.

A la derecha del símbolo ::= pueden darse varias producciones alternativas para el no terminal, separadas por barras verticales (|). Por ejemplo:

```
<incomplete message area> ::= <lost message area>   |   <found message area>
```

expresa que un <incomplete message area> es un <lost message area> o un <found message area>.

Pueden agruparse conjuntamente elementos sintácticos utilizando llaves ({ y }). Un grupo encerrado entre llaves puede contener una o más barras verticales, que indican elementos sintácticos alternativos. Por ejemplo:

```
<msc document body> ::= { <message sequence chart> | <msc diagram> }*
```

La repetición de grupos encerrados entre llaves se representa mediante un asterisco (*) o un signo más (+). Un asterisco indica que el grupo es facultativo y puede repetirse cualquier número de veces; un signo más expresa que el grupo tiene que estar presente y puede repetirse cualquier número de veces. El ejemplo anterior indica que un <msc document body> puede estar vacío pero puede también contener cualquier número de <message sequence chart> y/o <msc diagram>.

Si los elementos sintácticos están agrupados mediante corchetes ([y]), el grupo es facultativo. Por ejemplo:

```
<identifier> ::= [ <qualifier> ] <name>
```

expresa que un <identifier> puede, pero no tiene necesariamente que contener <qualifier>.

Los operadores de metalenguaje tienen el orden de presidencia que sigue: el operador que vincula más débil es el operador alternativo "|". A continuación viene la secuenciación de izquierda a derecha. Los operadores "+" y "*" vinculan más fuerte que la secuenciación y finalmente los corchetes "[...]" y las llaves "{ ... }" vinculan lo máximo.

11.2.2 Definición de la sintaxis concreta de bMSC

Se define la sintaxis concreta de un bMSC con restricciones de tiempo.

11.2.2.1 Reglas léxicas

Las reglas léxicas definen unidades léxicas. Las unidades léxicas son los símbolos terminales de la *sintaxis textual concreta.*

```
<left square bracket> ::=
                    [
<right square bracket> ::=
                    ]
<left curly bracket> ::=
                    {
<right curly bracket> ::=
                    }
<left open> ::=
                    (
<right open> ::=
                    )
<vertical line> ::=
                    |
<apostrophe> ::=
                    '
<comma> ::=
                    ,
<colon> ::=
                    :
<semicolon> ::=
                    ;
<full stop> ::=
                    .
<letter> ::=
                A | B | C | D | E | F | G | H | I | J | K | L | M
              | N | O | P | Q | R | S | T | U | V | W | X | Y | Z
              | a | b | c | d | e | f | g | h | i | j | k | l | m
              | n | o | p | q | r | s | t | u | v | w | x | y | z
<decimal digit> ::=
                 0 | 1 | 2 | 3 | 4| 5 | 6 | 7 | 8 | 9
<name> ::=
                {<letter> | <decimal digit> | <other character> | <full stop>}+
<other character> ::=
                ? | & | % | + | - | ! | = | * | " | < | > | _ |
                ^ | ~
<keyword> ::=          endmsc
                |      from
                |      in
                |      inf
                |      instance
                |      int_boundary
                |      is attached to
                |      msc
                |      mscdocument
                |      out
                |      reset
                |      set
```

```
                    |       text
                    |       timeout
                    |       to
```

11.2.2.2 bMSC

Un bMSC describe un flujo de mensajes entre instancias y especifica un comportamiento parcial de un sistema. Un bMSC se describe mediante instancias y eventos. Los eventos pueden ser de mensaje o de Timer. Además se pueden definir delay intervals y comentarios globales.

```
<textual msc document> ::=
                    mscdocument <mscdocument name> ; <message sequence chart>

<message sequence chart> ::=
                    msc <msc name> ; <msc body> endmsc ;

<msc body> ::=

                    [<instance parameter decl list>] <msc statement>*

<instance parameter decl list> ::=

                    <instance parameter decl>+

<instance parameter decl> ::=

                    <instance name> : instance ;

 <msc statement> ::=

                    <event definition> | <time interval> | <text definition>

<event definition> ::=
                    <instance name> : <instance event>

<instance event> ::=
                    {<message event> | <timer statement>}
```

11.2.2.3 Mensajes

Un mensaje dentro de un bMSC es una relación entre un envío y una recepción. La recepción proviene de una instancia y un envío se dirige hacia una instancia. La correspondencia entre el envío y recepción de mensaje debe definirse de manera unívoca.

En la representación textual, normalmente la correspondencia entre envío y recepción se desprende de la identificación del nombre del mensaje y de la especificación de la dirección.

```
<message event> ::=
                    <message output> | <message input>

<message output> ::=
                    out <msg identification> to <instance name> ;

<message input> ::=
                    in <msg identification> from <instance name> ;

<msg identification> ::=

                   <message name> [ , <message instance name> ]
```

11.2.2.4 Timers

En los bMSCs se puede especificar el seteo de un timer y una subsiguiente expiración o reseteo. La especificación del nombre de la instancia del timer es opcional, pero su duración es obligatoria. En el caso en que el nombre del timer no baste para una correspondencia unívoca, deberá emplearse el nombre de la instancia del timer.

```
<timer statement> ::=
                    <starttimer> | <stoptimer> | <timeout>

<starttimer> ::=
                    set <timer name> [, <timer instance name> ] <duration> ;

<duration> ::=
                    <left open> <durationlimit> <right open>
```

```
<durationlimit> ::=
                    <decimal digit>+ | inf

<stoptimer> ::=
                    stoptimer <timer name> [, <timer instance name> ] ;

<timeout> ::=
                    timeout <timer name> [, <timer instance name> ] ;
```

11.2.2.5 Delay intervals

Se utilizan delay intervals para definir restricciones de tiempo sobre la aparición de eventos: el retardo entre un par de eventos de un mismo bMSC puede ser restringido definiendo un límite mínimo o un límite máximo.

Un delay interval no implica que los eventos deban producirse. El cumplimiento de una restricción de tiempo es validado solamente si el evento relacionado con el final de ese delay interval sucede.

El intervalo de tiempo en un mensaje temporizado simboliza el retardo entre la salida del mensaje y la entrada del mismo. Además, los eventos salida de mensaje y entrada de mensaje pueden ser sometidos a restricciones de tiempo con respecto a otros eventos.

```
<time interval> ::=
                    [<interval label>] <bounded time>

<interval label> ::=
                    int_boundary <interval name> is attached to <messages>

<messages> ::=
                    { <msg output> | <msg input> } { <msg output> | <msg input> }

<bounded time> ::=
                    { <left open> | <left square bracket> }
                    [<decimal digit>+] , [<decimal digit>+]
                    { <right open> | <right square bracket> }

<msg output> ::=
                    out <msg identification>
<msg input> ::=
                    in <msg identification>
```

11.2.2.6 Texto

Existen definiciones de texto, que pueden utilizarse para comentarios globales.

```
<text definition> ::=
                      text <text name>;
```


DTO. DE COMPUTACION
INFOTECA
F.C.E. y N. - U.B.A

11.2.3 Definición de la sintaxis concreta de MSC-Graph

11.2.3.1 Reglas léxicas

Las reglas léxicas definen unidades léxicas. Las unidades léxicas son los símbolos terminales de la *sintaxis textual concreta*.

```
<left open> ::=
                    (
<right open> ::=
                    )
<vertical line> ::=
                    |
<comma> ::=
                    ,
<colon> ::=
                    :
<semicolon> ::=
                    ;
<full stop> ::=
                    .
<letter> ::=
                A | B | C | D | E | F | G | H | I | J | K | L | M
              | N | O | P | Q | R | S | T | U | V | W | X | Y | Z
              | a | b | c | d | e | f | g | h | i | j | k | l | m
              | n | o | p | q | r | s | t | u | v | w | x | y | z


<decimal digit> ::=
                0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<name> ::=
              {<letter> | <decimal digit> | <other character> | <full stop>}+
<other character> ::=
                ? | & | % | + | - | ! | = | * | " | < | > | _ |
                ^ | ~
<keyword> ::=
                      alt
                    | connect
                    | end
                    | endmsc
                    | expr
                    | msc
                    | mscdocument
                    | seq
                    | text
```

11.2.3.2 MSC-Graph

Los MSC-Graphs proporcionan medios para definir cómo puede combinarse un conjunto de bMSCs. Un MSC-Graph es un grafo dirigido en el que cada nodo puede ser:

i. un símbolo de comienzo (hay un solo símbolo de comienzo en cada MSC-Graph) (**expr**);
ii. un símbolo de terminación (**end**);
iii. una referencia a un bMSC; o
iv. una trama paralela: múltiples ejes salientes desde un mismo nodo. (**alt**)

Los ejes conectan los nodos del MSC-Graph e indican la secuenciación que es posible entre los nodos. La existencia de más de un eje saliente desde un nodo indica una alternativa.
Además se pueden incluir comentarios globales.

```
<textual msc document> ::=
                    mscdocument <mscdocument name> ; <message sequence chart>

<message sequence chart> ::=
                    msc <mscgraph name> ; <mscgraph> endmsc ;

<mscgraph> ::=
                    expr <mscgraph expression>

<mscgraph expression> ::=
                    <label name list> ; <node expression>* [<text definition>*]

<node expression> ::=
                    <label name> : { <node> seq ( <label name list> ) ; | end };

<label name list> ::=
                    <label name> { alt <label name> }*

<node> ::=
                    (<msc name>)

<text definition> ::=
                      text <text name>;
```

Nota de implementación:

A esta gramática definida para representar la sintaxis concreta, se le realizaron algunas modificaciones para poder utilizar la herramienta Telelogic Tau como editor de bMSCs y MSC-Graphs. Estas modificaciones se deben a la forma de generar los delay intervals con TAU (utilizando la sección de parámetros para los delay intervals de mensaje y comentario de eventos para los de velocidad de procesamiento) y al agregado de una constante cuando se define la duración de un Timer.

11.2.4 Definición de la sintaxis concreta de log de ejecución

11.2.4.1 Reglas léxicas

Las reglas léxicas definen unidades léxicas. Las unidades léxicas son los símbolos terminales de la *sintaxis textual concreta.*

```
<left open> ::=
                    (

<right open> ::=
                    )

<colon> ::=
                    :

<full stop> ::=
                    .

<minus> ::=
                    -

<numeral> ::=
                    #

<letter> ::=
                A | B | C | D | E | F | G | H | I | J | K | L | M
              | N | O | P | Q | R | S | T | U | V | W | X | Y | Z
              | a | b | c | d | e | f | g | h | i | j | k | l | m
              | n | o | p | q | r | s | t | u | v | w | x | y | z

<decimal digit> ::=
                0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<name> ::=
               <id>+

<id> ::=
              (<letter> | <decimal digit>)+ (<letter> | <decimal digit>)*
              <all character>*

<all character> ::=
              <other character> | <colon> | <full stop> | <minus>

<other character> ::=
              ? | & | % | + | ! | / | = | * | " | < | > | _ |
              ^ | ~

<date> ::=
               <four digits> <minus> <two digits> <minus> <two digits>

<time> ::=
               <two digits> <colon> <two digits> <colon> <two digits> <full stop>
               <nine digits>

<four digits> ::=
               <two digits> <two digits>

<two digits> ::=
               <decimal digit> <decimal digit>

<nine digits> ::=
               <decimal digit> <decimal digit> <decimal digit> [<decimal digit>]
               [<decimal digit>] [<decimal digit>] [<decimal digit>] [<decimal digit>]
               [<decimal digit>]

<keyword> ::=
               S | R
```

11.2.4.2 Log de ejecución

El log de ejecución está formado por la concatenación de pares de elementos: timestamp y el correspondiente envío o recepción de una etiqueta.

```
<execution log> ::=
              ([<comment>+] <log expression>)+

<log expression> ::=
              <timestamp> <label> [<comment>]

<label> ::=
              { S | R } <left open> <label name> <right open>

<timestamp> ::=
              <date> <time>

<comment> ::=
              <numeral> <comment name>
```