Universidad de Granada

Lenguajes y Sistemas Informáticos

Tesis Doctoral

MEDISTAM-RT: METODOLOGÍA DE DISEÑO Y ANÁLISIS DE SISTEMAS DE TIEMPO REAL

Kawtar Benghazi Akhlaki

Editor: Editorial de la Universidad de Granada
Autor: Kawtar Benghazi Akhlaki
D.L.: GR. 989-2009
ISBN: 978-84-692-0160-2

UNIVERSIDAD DE GRANADA

Fecha: **Deciembre 2008**

Autor: **Kawtar Benghazi Akhlaki**

Directores: **Manuel I. Capel Tuñón**

Juan A. Holgado Terriza

Título: **Medistam-RT: Metodología de Diseño y Análisis de Sistemas de Tiempo Real**

Departamento: **Lenguajes y Sistemas Informáticos**

Grado: **Ph.D.** Año: 2008

Firma de los directores:

A mi padre

Tabla de Contenidos

Índice de cuadros

Índice de figuras

Capítulo 1

Introducción

En este primer capítulo se explica la motivación que nos ha llevado a realizar este trabajo de tesis, exponiendo en primer lugar el problema que surge en la corrección de los sistemas de tiempo real, y la necesidad de llevar a cabo un proceso dirigido por modelos para el desarrollo del software, esto nos permite conseguir una especificación completa del sistema que incluya una descripción detallada de los requerimientos temporales, y además sea factible verificar si se satisface la corrección del sistema, e incluso los requerimientos temporales, antes de implementar el sistema. A continuación se discutirán qué soluciones admitidas hay para resolver dicha problemática, analizando propuestas aceptadas tanto desde el punto de vista académico como industrial en el estado del arte, y se motivará las razones que fundamentan la realización de este trabajo.

1.1. Antecedentes y motivación

Los sistemas de tiempo real constituyen un dominio de aplicación de gran interés tanto académico como industrial, ya que son fundamentales para el funcionamiento de sistemas críticos como los sistemas de control de aviones, sistemas de control de tráfico aéreo, sistemas de robótica, control de procesos, sistemas de defensa, sistemas de monitorización de pacientes, etc, o para

el funcionamiento de sistemas multimedia como sistemas de transmisión de video, sistemas de codificación o sistemas de videoconferencia entre otros. A diferencia de otro tipo de sistemas software, los sistemas de tiempo real son muy restrictivos en el sentido de que no sólo deben proporcionar una respuesta a las necesidades funcionales del sistema, sino que además deben hacerlo puntualmente en instantes o plazos de tiempos finitos [Kop97] [Ver94]. La consideración del tiempo en el diseño de un sistema informático dificulta su desarrollo e implementación, y obliga a tomar decisiones que restringen la capacidad de funcionamiento del sistema para que sea consistente con los plazos de tiempo que debe satisfacer, ya que en caso contrario el sistema fallará. Aspectos como la interacción, sincronización y coordinación de tareas, la interacción con el hardware, la estructuración del sistema en tareas, o la ordenación de la ejecución de dichas tareas, son algunos retos importantes a los que se tiene que enfrentar un desarrollador de software de tiempo real.

El aspecto esencial en el diseño de STR es el tiempo. Algunas de las cuestiones importantes que deben ser tratadas son: La obtención de la medida del tiempo o el tiempo de peor ejecución [Tov99], los mecanismos para el progreso del tiempo con relojes y temporizadores [LY03], la ordenación temporal en la ejecución de las distintas tareas del sistema de acuerdo al esquema de planificación [But97] [CDKM02], o la correcta especificación de las restricciones temporales del sistema [Fey97], [AM92]. De forma concreta, los requisitos de tiempo juegan un papel determinante en la corrección del sistema, y por tanto es necesario poder describir y razonar la información sobre su dimensión temporal.

La complejidad creciente de los STR hace cada vez más difícil la tarea del desarrollo. Los desarrolladores se enfrentan a la construcción de sistemas concurrentes que interaccionan entre ellos bajo restricciones de tiempo y que además en la mayoría de las veces están sometidos a un entorno externo impredecible. La necesidad de llevar un método estructurado para controlar la complejidad del desarrollo del software requiere la integración de técnicas de ingeniería del software dentro del proceso de desarrollo. De este modo podemos enfocar los esfuerzos de desarrollo en estudiar el sistema en el dominio del problema antes de realizar una implementación del mismo. Se han desarrollado una plétora de metodologías, marcos de desarrollo y métodos que han tratado de facilitar el desarrollo y mantenimiento de este tipo de sistemas, como

ROOM [SGW94], MAST [MGD01], COMET [Gom00], CODARTS [Gom93], el método de Shlaer-Mellor [SM96], OCTOPUS [AKZ96], SIMOO-RT [PBGN99] que constituye un entorno de desarrollo de sistemas de tiempo real, ROPES (*Rapid Object-Oriented Process for Embedded Systems*) [Dou99] que se propone como método completo de desarrollo de sistemas embebidos con requisitos temporales. Estos métodos se han definido para tratar problemas concretos en dominios de aplicación específicos como el dominio de avionica, sistemas embebidos o dispositivos de gran escala o de pequeña escala. Este tema sigue siendo un campo activo en la comunidad de tiempo real, en la que se está tratando de investigar y aplicar métodos de diseño avanzados que puedan facilitar el diseño de estos sistemas, y puedan posteriormente ser utilizados por la industria.

Una de las opciones para el desarrollo de los STR son las metodologías que utilizan paradigmas de desarrollo basado en modelos. En estos casos, los desarrolladores obtienen modelos acerca de los distintos aspectos del sistema en distintos niveles de abstracción y para las distintas etapas del proceso de desarrollo utilizando un lenguaje semiformal para la descripción de los elementos de modelado y sus relaciones.

Los métodos formales están considerados en la comunidad del software como un medio seguro que da alta confiabilidad a los sistemas del tiempo real. No obstante, estos métodos no llegan a tener una amplia aceptación en el desarrollo de software en la industria. La razón de este fracaso como ya ha indicado Richard Sharp en [Sha04], se debe principalmente a los requisitos de experiencia y habilidades matemáticas que deben satisfacer los desarrolladores para poder participar en un proyecto dentro de un marco formal, hay muy pocos programadores con un bagaje matemático aceptable, que se sientan cómodos aplicando los métodos formales, sobre todo en proyectos complejos. Anthony Hall, pionero de los métodos formales indicó " *There is big psychological resistance - people don't like mathematics. I don't know how to solve that problem*" [Hal90]. Esta fuerte resistencia a adoptar el uso de métodos formales requiere la aplicación de técnicas indirectos soportada por herramientas que faciliten por una parte el diseño de sistemas basados en el empleo de métodos formales, y por otra parte, proporcionan medios automáticos que permiten la verificación y validación de sistemas.

La combinación de métodos semi-formales y formales es una idea que ha

emergido en los años 90. La denominada aproximación mixta se ha considerado por muchos profesionales del dominio como un medio para beneficiarse de los dos métodos y al mismo tiempo superar la debilidad de los dos. Bruel en [BF98], indica "*the main objectives of integrated formal/informal approaches is to make formal methods easier to apply and to make informal methods more rigorous*". Aujla et al en [SBL94], muestra que las mismas técnicas formales se benefician de estar incluidos en un marco metodológico integrado. En efecto, estas ventajas abrieron una nueva área de investigación en la ingeniería del Software que sigue siendo muy activa en búsqueda de soluciones a los problemas que expone cada uno de estos tipos de métodos. Se han presentado diferentes técnicas, perfiles y metodologías basadas sobre aproximaciones mixtas, que se pueden contemplar bajo dos propósitos diferentes:

1. Dotar al lenguaje formal de una representación gráfica visual semiformal o formal que facilite la descripción de la especificación del sistema. En [DL91], se han definido técnicas para visualizar el lenguaje formal VDM [SM96] en modelos de análisis estructurado(SA). También, se ha definido una representación gráfica del lenguaje formal CSP+T en CSP-Jade [EBHC06]. Brooks propugna la visualización grtáfica [BP02] de los métodos formales y establece una notación gráfica para representar especificaciones en Timed CSP [Sch00] tambíen se han diseñado herramientas gráficas [JOGKL04] para el diseño de sistemas especificadas en CSP [Hoa78].

2. Dotar el lenguaje semi-formal de una semántica precisa para permitir la verificación y la validación formal de los sistemas. En [MJAJ02], se ha definido técnicas de transformación del método semi-formal STATEMATE [HP98] al lenguaje formal FNLOG [CSLO00] para garantizar la verificación de sistemas. Asimismo, múltiples trabajos integran el lenguaje semi-formal UML con métodos formales, existen propuestas para su combinación con el lenguaje Z [Spi92], Lógicas temporales (TL) [Kro87], CSP [Hoa85], π-calculus [Mil99], Circus (el lenguaje que combina CSP y Z) [SWC02], RT-LOTOS [CSLO00], Automatas temporizados [AD94].

1.2. Planteamiento del problema

UML es el lenguaje estándar en la industria. Su facilidad de uso, sus mecanismos de extensión y la diversidad de las herramientas que lo soportan, les está proporcionando mucha popularidad, incluso en el dominio de los sistemas de tiempo real. UML-RT representa la primera extensión de UML destinada a la adaptación de ese lenguaje para el diseño de los STR. Desde su aparición en el 1999, el perfil ha sido adoptado por muchos profesionales del dominio y soportado por una herramienta comprensible elaborada por IBM (Rational Rose Real-Time). Una prueba de éxito del UML-RT se manifiesta en la integración de muchos de sus aspectos en la reciente versión de UML (UML2.x). No obstante, el éxito de la aplicación de UML-RT en el dominio de tiempo real queda frenado por diferentes limitaciones que presenta el perfil en su definición.

UML-RT extiende el estándar UML con conceptos adicionales que proporcionan una ayuda significativa para el modelado de la arquitectura estática de sistemas interactivos. Sin embargo, el modelado de los aspectos dinámicos sufre de la ausencia de un modelo sintético que permita coordinar la comunicación entre diferentes componentes interactivos de un sistema de tiempo real. Además de esto, aunque UML-RT se ha elaborado para el tiempo real, no da soporte para tratar cuestiones de tiempo (las restricciones temporales, tiempo de ejecución, etc.), ni relaciones entre ellas. Otra crítica significativa de UML-RT en este contexto es su limitación en la representación de los requerimientos de los STR, dado que no da soporte a muchos fenómenos inherentes a este tipo de sistemas; por ejemplo, la representación del comportamiento no determinista o eventos simultáneos que puedan ocurrir en el entorno de los STR.

La semántica semiformal de UML representa una limitación muy considerable, sobre todo en los STR, donde una tarea de verificación de las propiedades (por ejemplo, seguridad, vivacidad, etc.) es imprescindible en el proceso de desarrollo de la aplicación. Es difícil realizar estas actividades, por no decir imposible, cuando la especificación del sistema está dada con modelos semiformales como UML o UML-RT. Adicionalmente, la semántica imprecisa de los modelos UML aumenta el riesgo de la aparición de ambigüedades en el curso del proceso de desarrollo en donde se utiliza el lenguaje, además, se

definen muy pocas reglas de verificación para comprobar la coherencia de un diagrama, y todavía menos para verificar la coherencia entre diagramas.

Observando los problemas que expone la aplicación del estándar al desarrollo de los STR y enfocándonos en la deficiencia en la expresión de los requisitos temporales. Consideramos que una aproximación que pueda lograr superar estas limitaciones debe:

1. *Extender la sintaxis de UML:* con conceptos y anotaciones de modelado relacionados con el tiempo real, con el fin de mejorar la capacidad expresiva de UML respecto al modelado del aspecto dinámico del sistema, adicionalmente, se debe establecer la relación entre estos conceptos, de manera que se pueda especificar requisitos temporales cualitativos y cuantitativos.

2. *Precisar la semántica de UML:* los requisitos temporales deben estar acompañados con una semántica formal, es decir, una semántica precisa y bien definida que descarta cualquier ambigüedad en la especificación de las propiedades temporales de un sistema, y que viene con principios y fundamentos matemáticos que permitan verificar y validar el sistema.

Con el fin de preservar la facilidad de uso de UML y reforzar su capacidad en manejar la complejidad de los STR, viene el interés en combinar el lenguaje con una notación formal que complementa la deficiencia de expresión de los aspectos de tiempo real cualitativos (sincronización, comunicación, coordinación) y los requisitos temporales.

De los métodos formales revisados y expuestos en el capítulo 2, nos decidimos por el uso del CSP+T, que es una extensión temporizada del algebra de procesos que parece ofrecer justo lo necesario para complementar la sintaxis y formalizar la semántica de UML. Nos pareció muy interesante la capacidad de expresión de CSP+T, que cubre los requisitos temporales, sobre todo la capacidad de restringir el tiempo de ejecución de eventos a un intervalo de tiempo específico (que puede ser absoluto o relativo) al tiempo de ejecución de eventos predecesores.

Adicionalmente, observamos que UML-RT y CSP+T encajan adecuadamente bajo una metodología que permita descomponer y recomponer el sistema utilizando mecanismos de los dos mundos. Y que ofrece una especificación

formal de STR que se puede usar como puente para pasar de un modelado semi-formal a generar código verificado correcto.

1.3. Objetivos de la tesis

El trabajo presentado en esta tesis se centra en las primeras fases del desarrollo de STR, tiene como objetivo principal la elaboración de un enfoque metodológico formal y fácil de usar, que combina la notación semiformal de UML-RT con los principios matemáticos del lenguaje formal. Para ello, hemos de lograr:

- Enriquecer la sintaxis de UML-RT con anotaciones de tiempo inspiradas de conceptos de CSP+T para permitir la especificación de los requisitos temporales desde la fase de diseño del sistema.
- Elaborar un método de diseño y análisis, bien definido que guía al desarrollador de un STR a modelar un sistema consistente. Representar un conjunto de reglas de mapeo que proponen una semántica precisa para las entidades de modelado de UML-RT sobre la base semántica que proporciona las leyes del álgebra de procesos CSP y CSP+T.
- Elaborar un método sistemático de derivación de la especificación formal de sistemas a partir de los modelos de UML-RT, aplicando las reglas de transformación antes mencionadas.
- Integrar los métodos anteriores con técnicas para establecer y verificar la consistencia de los modelos.
- Establecer las bases formales del lenguaje CSP+T para permitir especificar y verificar propiedades importantes de corrección de STR.
- Demostrar la utilidad práctica de la propuesta mediante su aplicación a un caso de estudio de interés en la industria.

Capítulo 2

Modelado de Sistemas de Tiempo Real

En este capítulo se define qué entendemos por un sistema de tiempo real, y las restricciones que añaden a un sistema. Posteriormente se señalan las distintas alternativas que actualmente se barajan para el modelado de los sistemas de tiempo real desde los enfoques semiformales basados en UML y los perfiles UML hasta los distintos enfoques formales, analizando los beneficios y problemas que proporcionan para el modelado.

2.1. Sistemas de tiempo real

Los sistemas de tiempo real son sistemas que deben reaccionar a ciertos estímulos en un tiempo finito y específico [BW01]. Es por tanto, un sistema informático que mantiene una relación interactiva y temporizada con su entorno [SR98]. La corrección de estos sistemas no depende solamente de los resultados lógicos, sino también del tiempo de respuesta en el que se producen esos resultados [SRS98]. Ejemplos comunes de los sistemas de tiempo real incluye los sistemas de control de aviones, sistemas de control de tráfico aéreo,

sistemas de robótica, control de procesos, aplicación de defensa, sistemas de monitorización de pacientes, etcétera.

Los sistemas de tiempo real imponen restricciones en el comportamiento temporizado de los sistemas. La corrección de un sistema de tiempo real no depende solamente del comportamiento funcional del sistema, lo que es esperable de cualquier sistema informático, sino también del comportamiento temporal del mismo. Es decir, las respuestas a los estímulos, en una aplicación se deben dar en intervalos de tiempo específicos. El grado de confiabilidad y cumplimiento de los requisitos temporales requeridos por estos sistemas los divide en dos categorías:

- **Estrictos (o duros)** en los que es imprescindible que la respuesta al estímulo se produzca dentro de un intervalo de tiempo específico. Es decir, estos sistemas imponen requisitos temporales estrictos, que son requerimientos que se deben de cumplir sin excepciones Cualquier excepción en la ejecución de un sistema estricto puede llevar a un fallo total del sistema, pudiendo provocar incluso daños materiales o personales. En estos casos, éstos sistemas se clasifican como sistemas críticos (*safety-critical*), es decir, sistemas en los que se debe prevenir situaciones no seguras del sistema antes de su implementación.

- **No estrictos (o suaves)** en los que un ocasional incumplimiento de uno o varios plazos de respuesta no altera el funcionamiento general del sistema. Las tareas de tiempo real no estrictas son ejecutadas tan rápido como sea posible, pero no están forzadas por tiempo límite absoluto En un incumplimiento de plazo de respuesta, el sistema va a seguir funcionando correctamente prestando los servicios para los que fue diseñado pero con una degradación de su calidad de servicio. Ejemplos típicos de los STR suaves son sistema de reservas de billetes, sistemas de comunicación como embedded commercial products (*television sets and videocassette recorders*).

Existe otra categoría, los sistemas de tiempo real firmes en los que, aunque se puede permitir el no cumplimiento de algún plazo de respuesta, no se tolera un incumplimiento continuo y duradero, de modo que debe tener un comportamiento firme con respecto al cumplimiento de los plazos de respuesta. El

ejemplo más característico de este tipo de sistemas son los sistemas multimedia.

Otra posible clasificación de los STR según Kopetz, es la siguiente [H.K91]:

- **Controlados por eventos:** son sistemas donde todas las acciones del sistema son causados por eventos como consecuencia de un cambio de estado observable o la ocurrencia de una interrupción de reloj.
- **Controlados por tiempo:** son sistemas donde todas las acciones del sistema son provocados por la progresión del tiempo. Estos sistemas tienen la habilidad de iniciar actividades después de que un intervalo de tiempo se cumpla, a partir de un punto de tiempo dado.

La mayoría de los STR son sistemas que mantienen interacciones continuas con el entorno exterior, a base de intercambio de estímulos. Eso requiere a los STR disponer de interfaces para adquirir los datos enviados del entorno externo y proveerle respuestas. Es importante aclarar que la corrección de la respuesta en el contexto del tiempo real no se refiere a la rapidez de procesamiento, como muchos pueden llegar a pensar, sino a la *puntualidad* de la respuesta dentro de un plazo de tiempo definido, con un límite de tiempo inferior y otro superior. Estos límites de tiempo pueden ser relativos al tiempo de ocurrencia de eventos predecesores a la respuesta. La especificación de estos plazos define las restricciones temporales del sistema. El cumplimiento de estas restricciones de tiempo hace muy complejo el desarrollo de los STR, y constituye una de las características más importantes de los STR.

Un STR depende fuertemente de su entorno, por lo que su comportamiento tiene carácter reactivo y dirigido por eventos (estímulos de entrada) a diferencia de lo que ocurre en los sistemas transformacionales. Estos sistemas reaccionan ante peticiones o estímulos del entorno en un intervalo de tiempo limitado y previsible. La respuesta a estos estímulos no siempre es la misma y depende del estado en el que se encuentra el sistema, es decir, la respuesta a una petición o estimulo del entorno, no depende solamente del estímulo recibido, pero también de la historia previa del sistema [Gom00].

Tanto el comportamiento funcional del sistema como su ritmo deben ser lo suficientemente deterministas para satisfacer la especificación del sistema [HS91]. Un sistema determinista produce las mismas secuencias de salida ante las mismas secuencias de entrada.

2.2. Tiempo y requisitos de Tiempo

Los requisitos de tiempo se suelen considerar como requisitos no funcionales del sistema, y representan restricciones temporales asignadas al funcionamiento de un STR. Con la especificación de las restricciones temporales en STR esperamos definir con precisión la puntualidad en las respuestas del sistema. Cuando el STR está compuesto por varias tareas en ejecución, aspectos como la concurrencia, la comunicación y la sincronización, pueden dificultar una captura precisa de los requisitos temporales dada las interacciones que se producen entre dichas tareas y la necesidad de ordenar la ejecución de dichas tareas, siendo necesario un análisis previo del sistema.

Podemos distinguir entre dos tipos de requisitos:

- **Requisitos de tiempo externos:** son requisitos temporales que se imponen a eventos externos [1]del sistema (eventos de comunicación del sistema con el entorno) para lo cual se deben cumplir algunos requerimientos no funcionales del sistema como el rendimiento, la velocidad de reacción o el plazo de respuesta (que puede ser determinista en algunas situaciones críticas).

- **Requisitos de tiempo internos:** son requisitos temporales que se impone a los eventos internos del sistema como, por ejemplo, notificación de algún componente del sistema. Estos requisitos pueden restringir el plazo de ocurrencia de los eventos; algunos ejemplos son que una acción debe ejecutarse en un plazo de tiempo definido, o el tiempo transcurrido entre dos eventos debe esta acotado con una cota máxima, o incluso los eventos del sistema deben ejecutarse en un orden preestablecido. La presencia de los requerimientos temporales complica la construcción de STR respecto a otros sistemas informáticos. La especificación rigurosa de las propiedades temporales de los sistemas ha sido un tema recurrente que ha preocupado a los investigadores de la comunidad de tiempo real [Hoa78], y por tanto, la especificación del tiempo es un aspecto que merece un tratamiento propio en el diseño de los STR [KDC96].

[1]Los eventos externos no incluyen en este contexto, los estímulos que se reciben del entorno, siendo este último incontrolable por el sistema.

2.3. Modelado de los sistemas en tiempo real

Las limitaciones que tiene nuestra capacidad mental para resolver problemas complejos nos lleva a simplificar el problema de manera iterativa hasta llegar a un grado de complejidad lo suficientemente adecuado para que pueda ser resuelto. En muchos casos la construcción de una solución correcta a un problema complejo no se puede llevar a cabo a través de modelos mentales, sino debe estar soportada por herramientas que permitan describir y representar los modelos mentales en modelos teóricos intuitivos, abriendo así la posibilidad de razonar y analizar el problema entero, y abordando sus conceptos más relevantes antes de proponer soluciones de implementación. Esta es la razón de que el modelado se convierta en una de las actividades más habituales en cualquier metodología moderna de la ingeniería del software, y que actualmente viene a definir la tendencia del desarrollo del software dirigido por modelos.

Un modelo se puede definir como una simplificación o una representación más sencilla de la realidad que nos permite quedarnos con los aspectos o abstracciones más relevantes de ese mundo real.

Para que un modelo pueda ser representable necesitamos un lenguaje de modelado o representación que proporcione una sintaxis y semántica bien definida a los modelos que se construyan con él. Esto permitirá que el modelo pueda ser comprendido por todos los miembros del equipo de desarrollo. Un modelo se puede describir con distinto grados de formalidad en función del lenguaje de representación manejado, y que a continuación podemos dividir en:

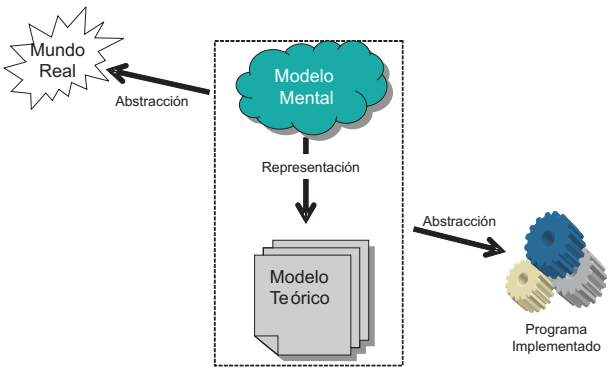


Figura 2.1: El rol de modelado

Un modelo se puede describir con distintos grados de formalidad como sigue:

- **Lenguajes no formales** como el lenguaje natural (el idioma español) en los que, aunque hay una sintaxis bien definida, la semántica en cambio depende fuertemente del contexto, siendo difícil abstraer con la suficiente precisión. A cambio proporcionan un mayor poder de expresión.
- **Lenguajes semiformales**, en los que, aunque se provee en un principio de una sintaxis y semántica bien definida, se utilizan elementos de representación lo más genérico posible para no perder capacidad de expresión, aún a costa de perder precisión. La ventaja es que de este modo pueden ser aplicados a distintos dominios de aplicación.
- **Lenguajes formales** en los que cada elemento del lenguaje está especificado con una sintaxis y semántica precisa basada en una descripción matemática o lógica. En estos casos, las restricciones que impone el lenguaje reduce la capacidad de expresión, pero a cambio se gana en precisión, y por tanto, facilita la verificación automática de los modelos que se construyan sobre el lenguaje.

Los modelos pueden ser más o menos formales en función del lenguaje de modelado que se ha utilizado. Aunque el lenguaje no fuerza la construcción de modelos, si es cierto que en muchos casos el lenguaje proporciona mecanismos o métodos para la realización de modelos, estando íntimamente ligados al lenguaje de modelado. Esto suele ocurrir, por ejemplo, con los lenguajes formales.

Con el aumento de la complejidad de los sistemas informáticos, se ha pasado de la preocupación que había por cuestiones de implementación, a dedicar más tiempo y esfuerzo al diseño y modelado del sistema. Esto se hace constatable desde finales de los años 70 con la aparición del diseño orientado a flujos de datos y diseño de datos estructurados, siendo estos los primeros métodos de diseño comprensibles y bien documentados. El diseño orientado a modelos se ha convertido en la actualidad en el proceso más aceptado para el desarrollo del software, y éste ha evolucionado desde las metodologías estructuradas a las orientadas a objetos, y las más recientes orientadas a aspectos.

En el caso del modelado de STR, aspectos como el cumplimiento de las restricciones temporales, la concurrencia, la criticidad, el acceso al hardware,

aumenta la complejidad del diseño de estos sistemas. La orientación a objetos ha proporcionado soluciones para mejorar el control de la complejidad, como la abstracción, el ocultamiento de la información, y también provee respuestas eficientes para la reusabilidad, extensibilidad mediante la encapsulación y herencia.

En las siguientes secciones vamos a profundizar en las posibilidades de modelado que ofrecen los lenguajes semiformales y formales más aceptados actualmente, especialmente para el tratamiento de STR, dado que son la base del método que se propone en este trabajo de tesis.

2.3.1. Modelado con lenguajes semiformales

Los lenguajes semiformales, a menudo, utilizan una notación gráfica que facilita la comprensión y visualización de los modelos que se desarrollan sobre ellos. Tradicionalmente, las metodologías de desarrollo de software proponen los principios, las heurísticas o reglas y los métodos que guían la construcción de modelos, así como el proceso a seguir para el desarrollo del software, señalando las actividades a realizar en cada etapa del desarrollo del software. Los lenguajes, en estos casos, se definían ligados a cada metodología o método para posibilitar la representación de los elementos que forman parte de los modelos.

Los primeros métodos de diseño, comprensibles y bien documentados que aparecieron, se fundamentaron en el diseño orientado a flujo de datos utilizando el principio de cohesión y acoplamiento para la estructuración del sistema software. Esté método se extendió para el modelado de los sistemas de tiempo real en el método SA/RT [HP87] en el año 1987.

La introducción del concepto de ocultamiento de información de David Parnas [Par79] fue el germen que cambió el proceso de construcción de sistemas software hacia lo que hoy se conoce como orientación a objetos. La capacidad de representar el mundo real en base a objetos y clases auto-contenidas, ha permitido al desarrollador preocuparse en primer lugar por el problema a resolver, y razonar sobre sus abstracciones, enfocándose en los conceptos relevantes del dominio del problema, para luego buscar una solución e implementación del mismo. Este nuevo enfoque de propósito general del desarrollo del software ha dado lugar a una plétora de métodos de análisis y diseño con

orientación a objetos, entre los cuales podemos citar como los más populares aplicado al dominio de los sistemas de tiempo real antes de la aparición de UML, ROOM [SGW94, SR98] y Octopus [AKZ96].

En ROOM podemos describir el STR en diferentes niveles de abstracción, pudiendo descomponer el modelado de un sistema en tres dimensiones: la estructura, el comportamiento y la herencia. El diseño con ROOM se basa en la definición del actor, que representa un objeto activo, es decir, un elemento que tiene asociado un hilo de ejecución propio, que le permite funcionar concurrentemente, con otros objetos activos. Los actores se comunican entre sí mediante paso de mensajes de acuerdo a un protocolo de comunicación. Cada actor tiene uno o varios puertos a través de los cuales interactúa con otros actores, un comportamiento descrito por un máquina de estados denominada Roomchart, y variables que se definen en términos de clase de datos. La reutilización de actores, protocolos y comportamiento se consigue a través de la herencia.

Es, a partir, de la aparición de UML, cuando el lenguaje de modelado se separa de la metodología que hace uso de él para la construcción de modelos. UML permitió unificar el lenguaje de modelado utilizado en las metodologías orientadas a objetos para el desarrollo de sistemas software. Al ser un lenguaje de modelado de propósito general, debe ser aplicable a cualquier dominio de aplicación y permitir el desarrollo de diferentes tipos de sistemas software desde sistemas de información a sistemas de tiempo real. Además esto facilita la lectura de los modelos y el desarrollo de herramientas que den soporte a dichas metodologías.

2.3.1.1. UML

UML representa un amplio compendio de formalismos de modelado que han dado un gran resultado al ser empleados en la descripción de sistemas complejos. En efecto, UML ha gozado de gran aceptación en el mundo empresarial y académico conviertiéndose en un estándar de facto siguiendo el paradigma de modelado orientado a objetos. Actualmente, UML es mantenido y desarrollado por el consorcio empresarial del Object Management Group (OMG).

Desde su especificación como lenguaje de modelado estándar del OMG, UML ha sido utilizado para describir un amplio número de sistemas críticos en

cuanto a tiempo y recursos. Sobre la base de esta experiencia, se ha alcanzado cierto consenso acerca de que, aun siendo una herramienta útil, UML carece de soporte para abordar algunos aspectos clave de particular interés para los diseñadores y desarrolladores de sistemas de tiempo real.

En concreto, la carencia de una noción cuantificable del tiempo y los recursos se ha considerado como un obstáculo a un uso más extendido en el dominio de los sistemas de tiempo real y/o sistemas empotrados. No obstante, dadas las capacidades de extensibilidad de UML, es posible abordar estas cuestiones de forma sencilla, sin añadir conceptos fundamentales de modelado a UML, ya que, las extensiones al lenguaje pueden aplicarse a nivel de metamodelo, conocido también como *heavyweight*.

OMG ha definido una jerarquía de metamodelado de 4 capas, de M3 a M0, donde cada capa se define como una instancia de la anterior: los conceptos en la capa Mn se definen en función de uno o diferentes conceptos de la capa Mn+1 que, a su vez, representa un metamodelo para Mn y un modelo de Mn-1.

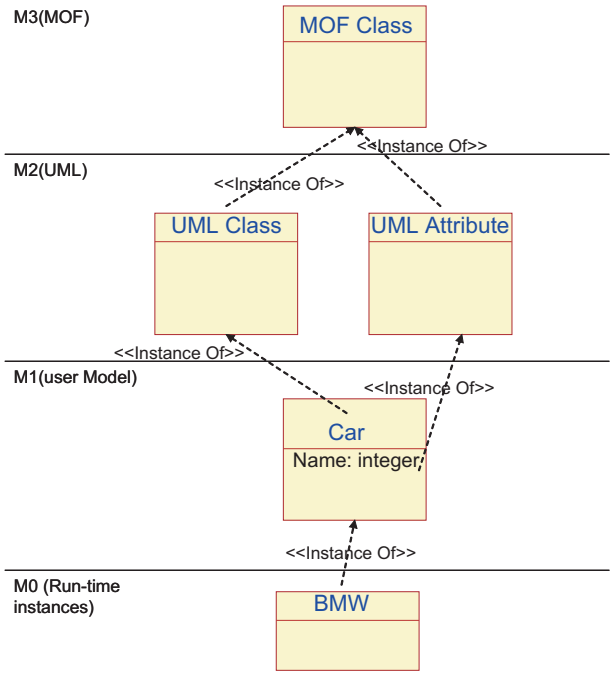


Figura 2.2: Ejemplo de un metamodelo de cuatro capas

1. La capa M3 representa la capa de metamodelado correspondiente a Meta Object Facility (MOF), que provee los mecanismos necesarios para definir una sintaxis abstracta con constructores básicos para describir los elementos comunes de cualquier metamodelo. MOF es utilizado como metamodelo para definir diferentes modelos tales como UML, Common Warehouse Metamodel (CWM) [Poo03], etc. Ejemplos de elementos de esta capa son *Meta-Clase*, *Meta-Atributo*, *Meta-Asociación* y *Meta-operaciones*.

2. La capa M1 define un lenguaje para describir un dominio semántico. En este nivel, los modelos del usuario son diseñados utilizando constructores de UML (elementos de la capa M2).

3. M0 es un modelo en tiempo de ejecución de los elementos del modelo de usuario definido en la capa M1. La figura 2.2 ejemplifica cada capa en la arquitectura del metamodelo, así como las relaciones entre los elementos de cada una de ellas.

Los conceptos y modelos de UML pueden agruparse en las siguientes áreas conceptuales [BRJ99, OMG07, OMG04a]:

- **Estructura estática.** Cualquier modelo preciso debe definir primero su universo, esto es, los conceptos claves de la aplicación, sus propiedades internas y las relaciones entre cada una de ellas. Este conjunto de construcciones proporcionan la estructura estática. Los conceptos de la aplicación son modelados como clases, cada una de las cuales describe un conjunto de objetos que almacenan información y se comunican para implementar un comportamiento. La información que almacena es modelada como atributos. La estructura estática se expresa con diagramas de clases y puede usarse para generar la mayoría de las declaraciones de estructuras de datos en un sistema.

- **Comportamiento dinámico**. Hay dos formas de modelar el comportamiento: una es mediante la historia de la vida de un objeto y cómo interactúa con el resto del mundo; la otra es a través de los patrones de comunicación de un conjunto de objetos conectados, es decir, a través de la forma en que interactúan los objetos entre sí. La visión de un objeto

aislado es la de una máquina de estados, que muestra la forma en la que el objeto responde a los eventos en función de su estado actual. La visión de la interacción de los objetos se representa mediante los enlaces entre objetos junto con el flujo de mensajes y los enlaces existentes entre ellos. Este punto de vista unifica la estructura de los datos, el flujo de control y el flujo de datos.

- **Construcciones de implementación.** Los modelos UML tienen significado para realizar el análisis lógico y para la implementación física de un sistema de software. Un componente es una parte física (hardware) o lógica (software) reemplazable de un sistema y es capaz de responder a las peticiones descritas por un conjunto de interfaces. Un nodo es un recurso computacional que define una localización durante la ejecución de un sistema. Puede contener componentes y objetos.

- **Organización del modelo.** La información del modelo debe ser dividida en piezas coherentes para que los equipos que desarrollan el sistema puedan trabajar en las diferentes partes de forma concurrente. Los paquetes son unidades organizativas, jerárquicas y de propósito general de los modelos de UML. Pueden usarse para almacenamiento, control de acceso, gestión de la configuración y construcción de bibliotecas que contengan fragmentos de código reutilizable.

En el cuadro 2.1, se resume el conjunto de diagramas que componen a UML.

2.3.1.2. Mecanismos de extensión de UML

El hecho que UML haya sido definido como un lenguaje de propósito general, permite su utilización para modelar sistemas de dominios de aplicaciones diversos. De hecho, desde su aparición, el lenguaje ha sido utilizado en diferentes dominios particulares como las telecomunicaciones, la gestión de información, el tiempo real, etc. A pesar de la riqueza de expresión de UML y su flexibilidad a la hora de modelar los diversos dominios de aplicación, siempre aparece la necesidad de disponer de un lenguaje más específico que ofrezca elementos de modelado que permitan describir los conceptos básicos y propios a un dominio

Área	**Vista**	**Diagramas**	**Conceptos Principales**
Estructural	Vista Estática	Diagramas de Clases	Clase, realización, generalización, dependencia, asociación, interfaz.
	Vista de diseño	Diagramas de Componentes	Componentes, interfaz, dependencia, realización, puertos, conectores.
		Diagramas de Estructura Compuesta	Puertos, conectores, interfaces.
	Vista de Despliegue	Diagramas de Despliegue	Nodo, componente, dependencia, localización.
Dinámica	Vista de Máquina de Estados	Diagramas de Estados	Estado, evento, transición, acción.
	Vista de actividad	Diagramas de Actividad	Estado, actividad, transición, decisión, división, unión.
	Vista de Casos de Uso	Diagramas de Casos de Uso	Caso de uso, actor, inclusión, asociación, extensión, generalización.
	Vista de interacción	Diagramas de Secuencia	Interacción, objeto, mensaje, activación.
		Diagramas de Colaboración	Colaboración, interacción, rol de colaboración, mensaje.
Gestión del modelo	Vista de Gestión de modelo	Diagramas de paquetes	Paquete, subsistema, modelo.

Cuadro 2.1: Diagramas que componen a UML

particular. Así, por ejemplo, el tiempo es un concepto básico y de fundamental interés en el modelado de sistemas de tiempo real o empotrados. UML ofrece varios mecanismos que permiten por un lado particularizar el uso del lenguaje y extenderlo para que pueda a aplicarse a dominios de aplicaciones específicos. Existen varias razones que motivan la adaptación del lenguaje de modelado UML para el modelado de sistemas [OMG04b]:

- Proporciona una terminología propia para plataformas o dominios de aplicación particulares.

- Define una sintaxis a construcciones que carecen de una notación propia (como es el caso con las acciones).

- Define nuevas anotaciones que sean más significativas e intuitivas para el dominio de aplicación que los símbolos ya existentes.
- Añaden semántica a construcciones que pueden estar definida de manera abstracta o imprecisa en el metamodelo.
- Añaden cierta semántica a construcciones que no existen en el metamodelo (por ejemplo, relojes, tiempo continuo, etc.).
- Añaden restricciones para restringir la forma de utilización del metamodelo y sus construcciones; por ejemplo, no permitir la ejecución paralela de ciertas acciones en una transición singular).
- Añadir información que puede ser útil a la hora de transformar un modelo a otro modelo o a código; por ejemplo, para definir las reglas de transformación de un modelo de diseño a código java.

UML tiene tres mecanismos para extender la capacidad del lenguaje: estereotipos, restricciones y valores etiquetados.

Estereotipos:

Permiten extender el vocabulario de UML con nuevos elementos de construcción que representan conceptos específicos a un dominio de problema o punto de vista.Estos elementos del Metamodelo pueden ser:

- Elementos estándares de UML.
- Una especialización de estos últimos, definida por el desarrollador para un dominio del problema especifico, que permita extender o cambiar la semántica y la sintaxis de un elemento de modelado, siempre que no contradiga la semántica de aquellos conceptos generales de los que se han derivado.

Gráficamente, un estereotipo se representa como un nombre entre comillas $\ll$ estereotipo $\gg$ y se sitúa junto al nombre del elemento que se está estereotipando. Opcionalmente, se puede asignar al estereotipo un icono que permita identificar visualmente las abstracciones véase la figura 2.3.

Figura 2.3: Estereotipo de reloj

Restricciones:

Representan un conjunto de propiedades que definen las condiciones que deben cumplir los elementos de modelado para que estén correctos. Las restricciones permiten extender la semántica de los bloques de construcción añadiendo reglas nuevas o modificando las existentes. Las restricciones se pueden expresar en un lenguaje especifico para la descripción de restricciones como el lenguaje OCL (Object Constraints Language)[WK99], un lenguaje de programación, un lenguaje matemático, o simplemente el lenguaje natural.

Valores etiquetados:

Representan valores de propiedades de los estereotipos aplicados en el los elementos del modelo. Los valores etiquetados, se pueden modelar solamente como atributos definidos para estereotipos. y se representan como una cadena encerrada entre llaves. Tal cadena incluye una etiqueta, un separador (el símbolo =), y un valor. {Versión = 3} es un ejemplo de cómo se representan los valores etiquetados.

El conjunto de estereotipos, restricciones y valores etiquetados que se han definido para representar los conceptos de un dominio específico de aplicación con el fin de personalizar el modelado y adaptarlos a dominios, plataformas y tecnologías particulares determina un perfil de UML. Así, es posible definir perfiles para el proceso de negocio, el tiempo real, la cualidad de servicio, etcétera.

Cómo la mayoría de los dominios involucran múltiples conceptos y muchos relacionados entre ellos, es conveniente agrupar sus correspondientes extensiones, de forma que se puedan aplicar conjuntamente. Por este motivo UML define un perfil en forma de paquete estereotipado «Profile» que especifica un

metamodelo para un dominio específico de interés. Cada elemento en el metamodelo se debe representar como estereotipo de una meta-clase prefinida de UML, tal como clases, asociaciones, componentes, etcétera. Por otra parte un perfil UML debe preservar la semántica estándar de los elementos UML, en el sentido de no contradecirla, aunque es permitido extenderla o especializarla todo lo necesario para adaptarla el dominio de interés.

La facilidad de uso de los mecanismos de extensión de UML se ha puesto de manifiesto con la amplia diversidad de perfiles que se han definido para propósitos específicos, incluso algunos de ellos adoptados y estandarizados por OMG. Entre ellos, distinguimos el Perfil UML para Planificación, Prestaciones, y Tiempo (*Scheduling, Performance, and Time*) (SPT) [OMG03], el perfil de UML para el modelado de calidad de servicio (*UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms*) (QoS) [OMG02] y el perfil para la experimentación (*Testing Profile*) (UTP) [BJ07]. También hay que mencionar la importancia de UML-RT [SR98], que si bien no satisface las condiciones para ser considerado un perfil de UML, ya que modifica en parte la semántica asociada a algunos bloques de modelado de UML, ha sido una de las propuestas pioneras que ha originado la creación de los perfiles en UML 2.0. Por otra parte, UML-RT supone la base metodológica que se ha utilizado para el desarrollo de nuestra propuesta.

2.3.1.3. UML-RT

UML-RT (UML for Real-Time) [SR98], es una extensión de UML para el modelado OO de sistemas de tiempo real, basada en el método ROOM.

UML-RT amplía UML con objetos activos estereotipados, denominados cápsulas, que representan componentes del sistema (actores en ROOM) añadiendoles variantes de diagramas de transición de estados para especificar su comportamiento dinámico. Los esfuerzos durante la definición de UML-RT han girado en torno a combinar las ventajas visuales de ambos enfoques [Her99]. En particular, los diagramas de actores de ROOM han sido incorporados a UML-RT con el nombre de diagramas de colaboración.

Para el análisis y el diseño de sistemas en tiempo real, UML-RT se basa en la noción de composición de elementos de modelado específicos: cápsulas, puertos, conectores, protocolos y roles de protocolos. La representación de

estos elementos de modelado son mostrados en la figura 2.4 y se describen a continuación.

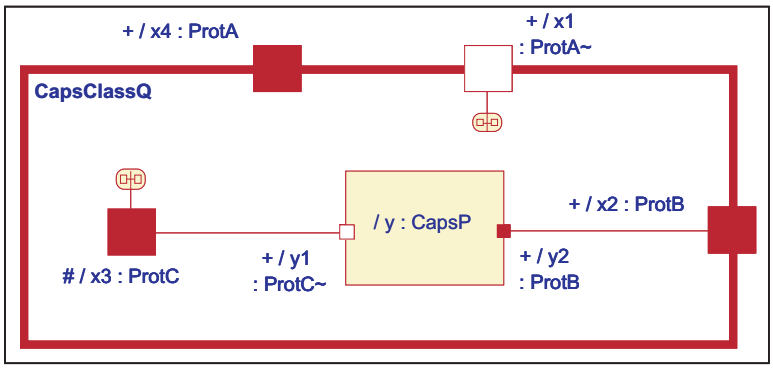


Figura 2.4: Diagrama de colaboración en UML-RT [SR98]

UML-RT agrega 4 nuevos bloques de construcción visual a UML, los cuales forman parte de un nuevo tipo de diagrama de colaboración para sistemas de tiempo real [FOW01, SR98]:

- *Cápsula.* Una cápsula es un estereotipo de una clase de UML con algunas características específicas. Las cápsulas se representan mediante construcciones rectangulares que exhiben una funcionalidad específica a través de una interfaz clara. El comportamiento de cada cápsula (su vida interna y cómo interactúa con el resto del mundo) se modela según un diagrama de transición de estados, que responde a eventos y genera acciones a través de puertos.

- *Puerto.* Un puerto representa un punto de interacción entre una cápsula y su ambiente que transmite señales entre el ambiente y la cápsula. La notación (símbolo) de un puerto es un cuadrado pequeño hueco. Se indica la visibilidad pública del puerto si el símbolo de éste está dibujado sobre el borde del rectángulo que denota a una cápsula en un diagrama de colaboración. Si el puerto está dentro del rectángulo, pasa a ser considerado como un puerto privado en la notación, ocultando su visibilidad al ambiente.

- *Conector.* Un conector es una abstracción de un canal de paso de mensajes que conecta dos o más puertos. Cada conector es determinado por

un protocolo que define las posibles interacciones que pueden ocurrir a través de ese conector.

- *Protocolo.* Un protocolo engloba el conjunto de comunicaciones válidas (intercambios de señal) entre dos o más cápsulas. Un protocolo está conformado por el conjunto de participantes (puertos), junto con los roles específicos que estos desempeñan en el protocolo.

El comportamiento interno de una cápsula se define usando *diagramas de estados* y la descripción del comportamiento combinado de varias cápsulas a través de diagramas de secuencia. Con estos diagramas, UML-RT permite definir modelos de sistemas en tiempo real con una semántica de ejecución correcta, que captura el comportamiento del sistema y soporta la simulación y el análisis del desempeño del sistema y de cada uno de sus componentes. A nivel de implementación UML-RT hereda todos los conceptos del análisis y diseño de componentes que define UML, pero UML-RT permite, además, la construcción de diagramas de componentes donde se incluyen los componentes de los diagramas nombrados anteriormente (cápsulas, puertos, protocolos y conectores).

2.3.1.4. SPT

OMG ha aprobado en septiembre de 2003 la adopción de la especificación final del SPT (llamado comúnmente UML de tiempo real) como especificación formal. El principal propósito del perfil es permitir un modelado avanzado de los sistemas de tiempo real, de manera que se puede hacer predicción cuantitativa concerniente el comportamiento tiempo real de una aplicación, a saber la planificación, rendimiento, y propiedades relacionadas al *timeliness*. El perfil pretende facilitar la comunicación de propuestas entre desarrolladores de forma estandarizada y también facultar la interoperatividad entre distintas herramientas de análisis y diseño.

El perfil SPT tiene una estructura de sub-modelos que agrupa en dominios los conceptos que éste aporta, de manera que se pueden utilizar o extender de forma independiente. Cada dominio tiene asociado un sub-perfil que define el conjunto de estereotipos que se proporcionan como uso directo a los modeladores, junto con los valores etiquetados propios de cada estereotipo

que dan valores a sus atributos y restringen su utilización. Precisamente, SPT está organizado en dos partes:

1. Un marco de trabajo que define un conjunto de conceptos genéricos, que representan una base para la definición de conceptos más elaborados para tratar lo relacionado con los asuntos de tiempo real.

2. Modelo de análisis del comportamiento tiempo real de las aplicaciones basadas en UML.

La primera parte consta de tres paquetes:

- *Modelado general de los recursos:* que define en términos muy generales los conceptos relacionados con los recursos y calidad de servicio.

- *Modelado general de la concurrencia:* define conceptos básicos para modelar la concurrencia y sincronización en las aplicaciones de sistemas de tiempo real.

- *Modelado General de tiempo:* define conceptos de tiempo y mecanismos relacionados con el tiempo, tal como los relojes , y los *timers*, etc.

El modelo conceptual que soporta el perfil SPT se muestra en la figura 2.6. El tiempo físico se percibe como una relación que impone un orden parcial entre los eventos [Sel03]. Este tiempo esta modelado como tiempo continuo y de progresión monótona de los instantes de tiempo y que tiene las siguientes propiedades [OMG05]:

- Conjunto totalmente ordenado, es decir, por cualquier dos elementos p y q pertenecientes al conjunto. p precede a q, o q precede a p.

- Conjunto de tiempo denso, es decir, que existe siempre un instante entre dos instantes de tiempo.

De ello, el tiempo físico es denso y progresa de manera monótona (y no como el tiempo virtual que puede regresar bajo algunas circunstancias). Las medidas de tiempo de un instante físico se representa con un valor especial llamado valor de tiempo. La duración es el tiempo expirado entre dos instantes de tiempo (instante de comienzo e instante de fin). Cada uno de estos instantes

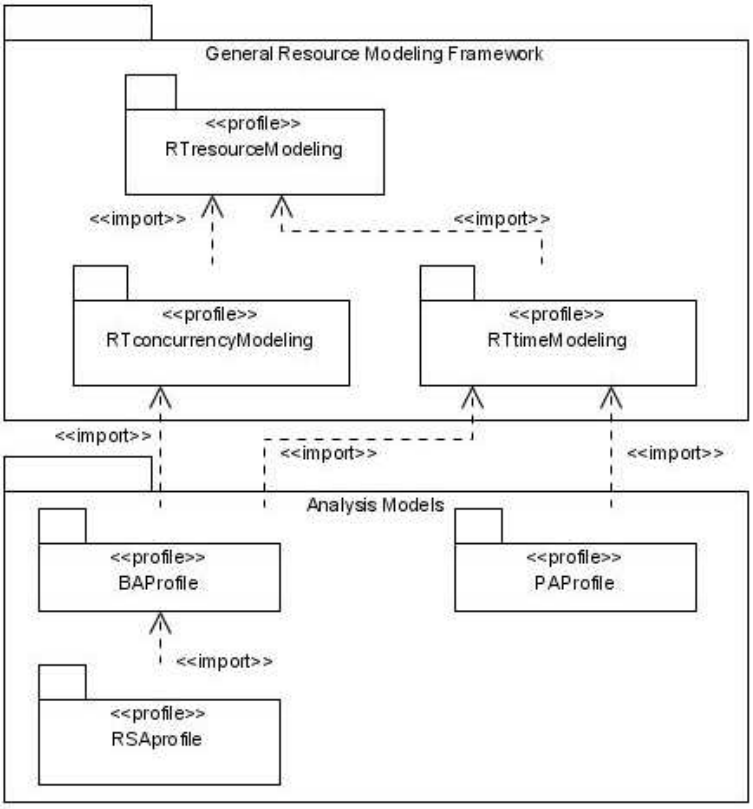


Figura 2.5: Estructura de SPT

se representa con un valor de tiempo. Así pues, el perfil introduce la noción del intervalo de tiempo [Sel03].

En adición al modelado de tiempo, el perfil SPT permite modelar dos mecanismos básicos para medir el tiempo: relojes y timers. Los timers son mecanismos asíncronos que generan un evento *timeout* cuando se produce la ocurrencia de un evento de tiempo específico. Este mecanismo es un recurso activo que detecta la expiración de un intervalo de tiempo o la duración. El reloj es también un recurso activo que genera de manera periódica interrupciones de reloj en sincronía con el progreso del tiempo físico, es decir, con la ocurrencia de los ticks de reloj.

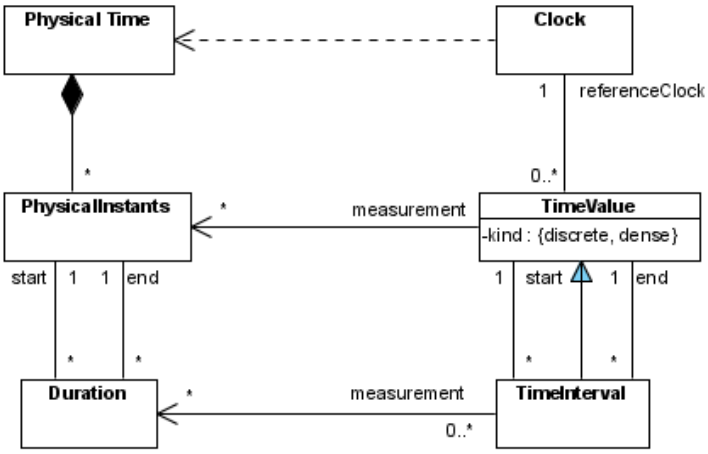


Figura 2.6: Conceptos básicos de modelado de tiempo

2.3.2. Especificación de sistemas con lenguajes formales

Con el crecimiento inevitable de los sistemas software en cuanto a funcionalidad, y la complejidad de desarrollo a la que se enfrenta los *Stakeholders* de estos sistemas, se aumenta la probabilidad de cometer errores en sus diferentes fases del desarrollo. Aún más, utilizando los métodos tradicionales, la detección de los errores sólo se podría realizar en etapas de depuración y mantenimiento del sistema. Estás etapas son muy tardías en el proceso de desarrollo, y pueden llevar a un aumento considerable en el coste del desarrollo, un retraso en la entrega del sistema, y lo más grave aún, la construcción de un sistema no válido dado que los métodos tradicionales no están dotados de mecanismos que garantizan la corrección del sistema. Ante este fenómeno el desafío más grande de la ingeniería del software es poder manejar la complejidad inherente de los sistemas para ofrecer sistemas confiables que funcionan de manera correcta bajo situaciones específicas. Los métodos formales permiten detectar ambigüedades, problemas e inconsistencias.

Cuando se aplican en etapas tempranas del desarrollo del software pueden detectar fallos en la concepción del sistema que de otra manera no se podrían revelar hasta llegar a las etapas de pruebas que son muy costosas.

Cuando se aplican más tarde en el proceso de desarrollo, pueden ayudar a

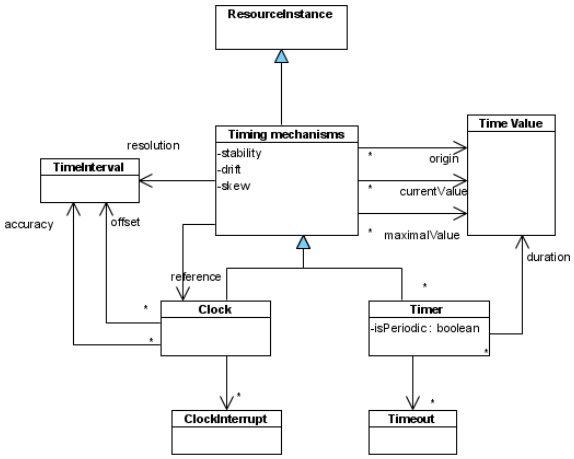


Figura 2.7: Mecanismos para el modelado de tiempo

determinar la corrección de la implementación de un sistema y la equivalencia entre diferentes implementaciones. La utilización de los métodos formales puede ser el camino para garantizar el desarrollo de software correcto. Sin embargo, aún existen muchos inconvenientes para la correcta implantación de los métodos formales como una metodología de desarrollo de software comúnmente aceptada en ingeniería de software.

Los métodos formales constituyen un enfoque analítico para la especificación, diseño, validación y verificación de los sistemas software y hardware. La fuerza de estos métodos se manifiesta en la precisión y la rigurosidad en las que sus modelos se encuentran basados, con fundamentos en sólidos principios matemáticos que permiten definir con precisión y sin ambigüedades las necesidades del sistema. Los modelos en los métodos formales se representan mediante descripciones matemáticas o lógicas utilizando un lenguaje formal, a partir de los cuales podemos obtener la especificación del sistema; de ahí, que en ocasiones a los lenguajes formales se les denomine lenguajes de especificación. Gracias a estos fundamentos la especificación del sistema puede ser verificada mediante el cumplimiento de propiedades derivadas de la especificación.

Estas propiedades del sistema pueden incluir el comportamiento funcional, el comportamiento temporal y también la estructura interna del sistema. Hasta

ahora la aplicación de los métodos formales ha tenido más éxito en la especificación y la verificación de las propiedades comportamentales. Los métodos formales suelen ser de propósitos específicos; esto justifica la inmensa variedad de métodos formales existentes en la literatura.
Podemos hacer una clasificación de los métodos formales atendiendo a la capacidad para **la especificación** de sistemas en función del lenguaje utilizado para la especificación y modelado de los sistemas:

- **Lenguajes de especificación para sistemas secuenciales**. La especificación del sistema se hace mediante la descripción explicita de estados y operaciones que transforman el sistema de un estado a otro. Este tipo de lenguajes está destinado a la especificación de los sistemas secuenciales, y por tanto no dispone de ningún mecanismo que permita representar la concurrencia es posible. En esta categoría de métodos formales encontramos los lenguajes [Spi92], VDM [Jon90], y el método B [Abr96].

- **Lenguajes de especificación para sistemas concurrentes.** En estos casos, el comportamiento de los sistemas se define en términos de secuencias, árboles u orden parcial de los eventos. Ejemplos de estos lenguajes son CSP [Hoa85], Π-Calculus [MPW92], Statechart [Har87], Lógica Temporal [Pnu77a], etc.

- **Lenguajes de especificación para sistema temporales**. En general, representan extensión de las anteriores clases de lenguajes para describir el comportamiento temporizado de los sistemas. El comportamiento de los sistemas se representan con modelos temporizados o secuencias de eventos temporizados. Dentro de esta clase se clasifica extensiones de tiempo de algebra de procesos, timed CSP [Sch00], Timed CCS [Yi90], ATP [NJ94].

Otra clasificación de los métodos formales basada en las técnicas utilizadas en la descripción de sistemas, los divide en cinco tipos [YW03]: Lenguajes basados sobre modelos, Lenguajes algebraicos, lógica temporal, álgebra de procesos y lenguajes basados en Redes. Nos interesamos particularmente, en los tres últimos tipos, ya que permiten la especificación del comportamiento de sistemas.

A continuación, vamos a profundizar en los aspectos relacionados con la concurrencia y el tiempo de estos lenguajes de especificación empleados tradicionalmente.

2.3.2.1. Lenguajes basados en lógica temporal

Entre otras aplicaciones, la lógica temporal puede emplearse para especificar cómo va a ser el comportamiento de un sistema con el paso del tiempo. Para ello, a nivel sintáctico se define una serie de operadores con capacidad para expresar conceptos de tiempo y diferenciar entre el pasado, el futuro y el presente; como, por ejemplo, el operador clásico $\Box$(*siempre*), y el operador $\diamond$ *(alguna vez)*, *O (next)*. Estos operadores permiten especificar propiedades cualitativas, sin embargo, no tienen capacidad para expresar propiedades cuantitativas, es decir, restricciones de tiempo real.

La lógica temporal fue estudiada por primera vez, y presentada como herramienta de argumentación filosófica que permitía la interpretación del paso del tiempo, en el año 1977. Así, Arthur Prior en [Pnu77a], propuso el uso de esta lógica para la especificación de sistemas reactivos.

Las lógicas temporales pueden clasificarse en *lógicas temporales lineales* y *lógicas temporales ramificadas*:

- Las lógicas temporales lineales, como LTL [Pnu77b] o PTL [DPSS80], permiten expresar propiedades relacionadas con caminos [2] individuales de un programa. La sintaxis de este tipo de lógica incluye distintos operadores como el operador $\Box$(*siempre*), y el operador $\diamond$ *alguna vez*, *O* (next), *U* (Until). Por ejemplo, la propiedad *siempre que el semáforo esté en rojo (R), entonces en algún momento en el futuro tiene que estar en verde (V)*, se puede expresar como:

  $$\Box(R \rightarrow \diamond V)$$

  Esta propiedad especifica que para cualquier posible ejecución del sistema en la que se produzca el estímulo R, en algún momento posterior el sistema llegará a un estado en el que producirá V.

[2] Una secuencia infinita $Q = q_1 q_2 q_3 \ldots$ es un camino en un autómata si $\forall i : q_i$ es un estado y existe un arco de q_i a $qi + 1$.

- Las lógicas temporales ramificadas como CTL [], por el contrario, permiten expresar propiedades relacionadas con árboles de ejecución de un programa. Las propiedades expresadas en esta lógica incluyen operadores para expresar conceptos como *inevitablemente* (para todo futuro, siempre), *potencialmente* (para algún futuro, a veces), *inevitablemente* (para todo futuro, a veces), *invariablemente* (para todo futuro, siempre). Por ejemplo, la propiedad *siempre que el semáforo esté en rojo (R), entonces en algún momento en el futuro tiene que estar en verde (V)*, se puede expresar con la siguiente fórmula:

$$\forall\Box(R \rightarrow \exists \diamond V)$$

Esta fórmula requiere que en cualquier posible ejecución del sistema en la que se produce el estímulo R, existe un potencial progreso del sistema donde, en algún momento futuro, el sistema estará en un estado en el que se disparará el evento V.

A la hora de extender la lógica temporal con objeto de poder especificar propiedades de tiempo real en un sistema, se han usado dos enfoques principales [AD94]: **relojes ocultos** y **relojes explícitos**.

- Las extensiones basadas sobre *relojes ocultos*, utilizan el método de los *operadores temporales acotados (bounded operator)*, que consiste en reemplazar los operadores de la lógica temporal por operadores acotados en el tiempo. Por ejemplo, el operador $\diamond_{[2,5]}$, se interpreta como *algunas vez entre los instantes 2 y 5*. Entre las lógicas temporales basadas en el enfoque de relojes ocultos se pueden destacar *Metric Interval Temporal Logic* (MITL) [AFH91], *Real time Computation Tree Logic* (RTCTL) [EMSS91].

- Las extensiones basadas en *relojes explícitos* utilizan dos métodos, a saber, (*cuantificación congelada* y *una variable reloj explícita*):

    1. La cuantificación congelada: Asigna a variables al tiempo de estados particulares. La fórmula $x.(\phi(x))$, liga el operador de congelación x al tiempo del contexto temporal actual, y se cumple para el tiempo t si $\phi(t)$ se satisface. De esta forma, una propiedad como *cada petición p está seguida por una respuesta q, dentro de 10 unidaddes de*

tiempo se definiría con la siguiente fórmula:

$$\Box x.(p \rightarrow \diamond y.(q \wedge y \leq x + 10))$$

Y se interpreta de tal forma que, cuando se produce una petición p, y la variable x está congelada en el instante actual, la petición está seguida de una respuesta q, en un instante de tiempo y, tal que $y \leq 10$. Entre las lógicas basadas en la cuantificación congelada podemos encontrar TPTL[AD94] y TCTL[AH93].

2. variable de tiempo explícita [ELA90]: representa una variable dinámica T, que está ligada a un reloj y una cuantificación de las variables temporales. Así, la propiedad anterior se podría expresar como:

$$\Box x.(p \wedge (T = x))) \rightarrow \diamond(q \rightarrow \wedge(T \leq x + 10))$$

donde la variable x, toma sus valores del tiempo de los estados en que se satisface p. Ejemplos de lógicas basadas en una *variable de reloj explícita* son *Real Time Temporal Logic* (RTTL) [MP92], y *eXplicit Clock Temporal Logic* (XCTL)[ELA90].

2.3.2.2. Lenguajes basados en álgebra de procesos

El término *álgebra de procesos* se utiliza con diferentes significados. Por un lado, la palabra *proceso* se refiere al comportamiento[3]. Por su parte, el álgebra es la rama de las matemáticas en la cual las operaciones aritméticas son generalizadas empleando números, letras y signos. Por esta razón, el término *álgebra de procesos* aplicado a la especificación del comportamiento de un sistema indica una aproximación algebraica/axiomática para describir y tratar dicho comportamiento [Bae05].

Con carácter general, las álgebras de procesos como *Comunicating Sequencial Processes* (CSP) [Hoa85], *Calculus of Communicating Systems*(CCS) [Mil89] o *Language of Temporal Ordering Specification* (LOTOS) [BB87], están enfocadas hacia el tratamiento de sistemas complejos y concurrentes, por medio de aproximaciones basadas en *la teoría de la concurrencia*[4]. Asimismo,

[3]el comportamiento, a su vez, significa el conjunto de acciones y eventos que realiza el sistema, y su potencial orden de ejecución, así como, algunos otros aspectos relacionados con la propia ejecución como su temporización o las probabilidad de que se produzca cada ejecución alternativa.

[4]Una teoría de sistemas paralelos, interactivos y/o distributivos

proporcionan medios algebraicos para describir y especificar estos sistemas, incluyendo medios para tratar la *composición paralela*, la *composición alternativa* –elecciones– de procesos, así como, la *composición secuencial* y *ocultamiento* de eventos. De esta manera, estos formalismos permiten construir sistemas complejos a partir de procesos simples. En general, los procesos más simples suelen ser especificados con eventos, y los procesos complejos, se derivan de los más simples utilizando operaciones de composición, sincronización y ocultamiento.

En la literatura relacionada, se han definido diferentes extensiones de álgebras de procesos que permiten especificar restricciones temporales. Entre ellas destacamos Temporal CCS [MT90], Timed CSP [Sch00], CSP+T [Ž94],

RT-LOTOS

Es una extensión de LOTOS basada en un dominio de tiempo denso, y donde el tiempo es introducido con dos operadores:

1. operador de retraso determinista «$delay(d),\ P$», para especificar que un proceso P espera un retraso d.
2. operador de retrasos nodeterminista «$latency(l),\ P$», para especificar que un proceso puede elegir un retraso aleatorio dentro del intervalo $[0, l]$, antes de comportarse como el proceso P.

Timed CSP

Es una extensión de CSP con un modelo temporal denso que utiliza un reloj global. Esta extensión introduce tres operadores principales para la medida del tiempo:

1. Operador **timeout**: $P \rhd^t Q$, que pasa el control del proceso P al proceso Q si P no se involucra en ninguna comunicación antes del tiempo t.
2. Operador **prefijo con tiempo**: $a \rightarrow^t Q$, que pasa a comportarse como P en t unidades de tiempo tras la ocurrencia del evento a.
3. Operador de **retraso**: $Wait\ t \rightarrow P$, que retrasa la ejecución del proceso P con un tiempo t.

2.3.2.3. Métodos formales basados en grafos

Los métodos formales basados en grafos son métodos con anotación gráfica dotada de una semántica formal. En este tipo de métodos encontramos, Redes de Petri coloreadas, Statecharts [Har87], *specificaction and description language*(SDL) [ITU].

SDL

Es un lenguaje formal basado en los *Automatas a Estados Finitos Comunicantes* (EFSM) [HKU02, LHH03] normalizado por la ITU-T en 1976. Muy utilizado en la comunidad de la telecomunicación. SDL, se ha aplicado también, con éxito, en muchos otros dominios, tal que, los sistemas embebidos. SDL tiene dos modos equivalentes de especificación, SDL textual y SDL gráfico. Una especificación en SDL se estructura en tres niveles de abstracción:

1. El sistema: define los limites entre el sistema y su entorno.

2. El bloque: permite dividir y estructurar jerárquicamente el sistema para manejar su complejidad. Consta de uno o más proceso (o bloques). Los bloques se conectan entre ello o con el entorno mediante *canales*.

3. El proceso: reagrupa las acciones efectuadas. Un proceso es una *máquina de estado finita* (FSM), que comunican mediante mensajes (señales) entre ellas y con el entorno. todas las máquinas funcionan en paralelo.

El comportamiento de un proceso se define mediante un diagrama de estado. La recepción de un estímulo provoca un cambio de estado del proceso. Este estímulo puede ser una señal de entrada o señales temporales. Existen dos principales mecanismos para tratar el tiempo en SDL:

1. Reloj Global: se define la función *now*, que retorna el tiempo actual del reloj.

2. *timer*: se define a nivel de proceso. Durante la inicialización del *timer* se le asocia un valor de expiración, que puede ser relativo, o absoluto. Para ello, se definen dos operaciones:

- **Set**(**t**, **temp**), activa el *timer* que expira en t ticks de reloj, y luego manda una señal *temp* al proceso propietario del *timer*.
- **RESET**(**temp**), desactiva el *timer temp*, antes de que expire.

Cada proceso dispone de una cola de espera, *First in First out FIFO*, donde se almacenan las señales no tratadas. Este mecanismo permite establecer prioridades en el tratamiento de las señales, para ello, permite implícitamente, tomar en cuenta restricciones temporales de la aplicación.

2.4. Verificación de sistemas

Para los procesos de **verificación,** básicamente existen dos tipos de técnicas de análisis formal: una basada en la demostración de teoremas (*Theorem proving*), y otra basada en la evaluación de modelos (*Model-checking*).

- **Theorem proving:** en esta técnica se describe en términos de fórmulas lógicas [CW96] el modelo del sistema y las propiedades que se desea que cumpla dicho sistema. A partir del conjunto de fórmulas que representan el sistema se demuestran las propiedades, utilizando para ello, mecanismos de deducción y substitución. En diferentes etapas de esta prueba, las fórmulas representando el sistema se reescriben respetando un conjunto de reglas hasta la obtención de las propiedades.
- **Model checking:** esta técnica está basada en modelos que describen el comportamiento de manera matemática precisa y no ambigua. Estos modelos se acompañan de algoritmos que exploran sistemáticamente los estados [5] del modelo. De esta manera, se muestra que ciertas propiedades se cumplen en el sistema.

[5] Los estados representan potenciales escenarios del sistema

Capítulo 3

Modelo Semántico de CSP y CSP+T

3.1. Introducción

Primeramente, en este capítulo presentamos el lenguaje CSP mediante la descripción de la sintaxis y semántica denotacional del mismo. Posteriormente, se introduce el lenguaje CSP+T, para el que se ha definido una semántica denotacional. Asimismo, se describe cómo especificar propiedades de sistemas de tiempo real en los modelos denotacionales establecidos para el lenguaje CSP+T.

3.2. Comunicación de procesos secuenciales

El formalismo *Comunicación de Procesos Secuenciales* (CSP) [Hoa78] fue propuesto por Tony Hoare entre 1975 y 1985 como un modelo teórico de programación destinado a la especificación y análisis de sistemas interactivos y

concurrentes [Sch00], cuyos requerimientos conciernen principalmente las interacciones entre el componente y su entorno.

CSP constituye uns notación de programación cómoda para llevar a cabo la programación de multiprocesadores de memoria distribuida y posee ventajas adicionales, tales como propiciar la capacidad de verificación y la portabilidad entre diferentes arquitecturas de los programas que se obtienen utilizando un lenguaje que tenga como base a la citada notación. CSP está basado en un cálculo teórico que proporciona un conjunto de términos sintácticos que representan a clases de procesos. La notación de programación CSP está basada en órdenes que especifican el comportamiento de un componente que lo ejecuta y su resultado puede ser tener éxito o fallar. Si la ejecución de una única orden tiene éxito, puede tener efecto sobre el estado interno del dispositivo (p.ej, la orden de asignación puede modificar los valores de las variables que representan el estado del dispositivo), o bien en el entorno del dispositivo (orden de salida), o bien en ambos (orden que representa la recepción de un mensaje y su asignación a una variable objetivo del proceso receptor).

3.2.1. Procesos y eventos

En el marco conceptual de CSP, los componentes y procesos se consideran como entidades independientes y *auto-contenidos* que interaccionan con el entorno externo a través de interfaces. Asimismo, CSP provee un marco *composicional* que permite razonar sobre la dinámica de los sistemas; la composición de dos procesos representando sistemas independientes da lugar a un proceso representando otro sistema independiente y *auto-contenido* que a su vez interacciona con el exterior sólo a través de interfaces particulares.

Para la definición de los procesos como cajas negras que interaccionan con el mundo exterior a través de interfaces, CSP centra la información relevante de un proceso, en el comportamiento en sus interfaces, estas son las comunicaciones externas del proceso con el exterior.

Por consiguiente, CSP representa:

- Las interfaces como conjunto de eventos, llamado ***alfabeto de comunicación*** del proceso (el alfabeto de comunicación de un proceso P se denota αP o $\alpha(P)$), es decir, los eventos en los cuáles el proceso se puede

involucrar. Esta representación de las interfaces se considera como una descripción estática del del potencial comunicación proceso.

- El comportamiento de un sistema (proceso) como conjunto de posibles ejecuciones; cada ejecución incluye una determinada secuencia de eventos pertenecientes al alfabeto de comunicación del proceso.

El modelo de comunicación del CSP se basa sobre dos suposiciones fundamentales [RHB97]:

- Las comunicaciones son instantáneas: la ocurrencia de los eventos se da en un momento unitario.
- Una comunicación sólo ocurre cuando tanto el proceso como el entorno lo permitan. Asimismo, un evento tiene efecto sólo cuando el entorno y el proceso convienen en su ejecución.

3.2.2. Operadores de CSP

Los términos CSP se construyen conforme a la siguiente gramática:

$$P ::= \quad STOP \mid SKIP \mid a \rightarrow P \mid a : A \rightarrow P(a) \mid P \Box Q \mid P \sqcap Q \mid$$

$$P; Q \mid P \underset{A}{\|} Q \mid P \|\| Q \mid P \setminus A \mid f(P) \mid \mu X.$$

Estos términos tienen la siguiente interpretación intuitiva:

Interbloqueo

$STOP$ es el *interbloqueo*, un proceso estable que sólo puede dejar pasar el tiempo.

Terminación satisfactoria

$SKIP$ es el proceso de terminación satisfactoria. De manera intuitiva, corresponde al proceso $\surd \rightarrow STOP$, que es capaz de terminar inmediatamente (en cualquier instante de tiempo), comunicando el evento de terminación especial ($\surd$) a su entorno.

Prefijado de eventos

$a \rightarrow P$ inicialmente ofrece en cualquier momento involucrarse en el evento a para pasar a comportarse como P, esto es, el proceso P no se activa hasta que se dé el evento a.

Prefijado general

El proceso de prefijado general $a : A \rightarrow P(a)$ está preparado inicialmente para involucrarse en cualquier evento $a \in A$, a elección del entorno, para después comportarse como $P(a)$, es decir, se elige el comportamiento del proceso en función de la elección del evento. $P(a)$ corresponde a $STOP$ cuando $A = \emptyset$.

Elección

Para poder representar un comportamiento alternativo, el lenguaje ofrece dos formas de hacerlo, *la elección externa*, ddonde la selección se hace en el entorno, y *la elección interna,* donde esta selección la hace el proceso mismo.

- **Elección Externa**: $P \Box Q$, representa un proceso que puede comportarse como P o como Q, según lo decida el entorno. Está decisión se toma a partir del primer evento visible (y no antes). Por ejemplo,

  $$(a \rightarrow P \Box b \rightarrow Q)$$

  es el proceso que va a comunicar los dos eventos iniciales a y b, para posteriormente comportarse bien como, P o bien como, Q , depende de los eventos iniciales que el entorno elige comunicar. La elección se resuelve de manera no determinista, si ambos eventos, a y b, se comunican simultáneamente, o bien, el evento inicial es posible para ambos P y Q. Por ejemplo: si R= $(a \rightarrow P) \Box (a \rightarrow Q)$, la elección del evento a por el entorno de R, no determina por qué camino (P o Q) deberá transitar el proceso R.

  CSP ofrece otro operador "|", al que se le denomina **alternativa etiquetada**, y que tiene un significado idéntico a $\Box$; la diferencia es que este último admite términos que el primero no. Estos son:

A ambos lados de | deben estar explícitas las dos alternativas que se ofrecen al entorno, y éstas deben ser diferentes. Por ejemplo, los términos $P \,|\, Q$ y $(a \to P) \,|\, (a \to Q)$ no estaán permitidos en CSP.

- **Elección Interna**: $P \sqcap Q$ denota un proceso que se va a comportar como P o Q. Sin embargo, es independiente del entorno, es decir, no se le permite al entorno ningún control sobre la seleccíon del evento. Por ejemplo ,el proceso

$$(a \to P) \sqcap (b \to Q)$$

puede comportarse como P o como Q. El proceso puede rechazar a o b, y es solamente obligado a comunicar, si el entorno le ofrece ambos eventos a y b, simultáneamente.

Composición secuencial

$P; Q$ corresponde a una composición secuencial de P y Q. Denota el proceso que se comporta como P hasta que P elija terminar, y empieza a comportarse como Q.

Como podemos prever, $\forall P$, $SKIP; P = P$ y $STOP; P = STOP$. La primera ecuación afirma que como $SKIP$ no hace nada excepto pasar el control a P, el comportamiento resultado no se diferencia del comportamiento de P. En la segunda ecuación, el proceso de interbloqueo $STOP$ no indica terminación; así, el control no se va a pasar nunca al proceso P, y el resultado es equivalente a $STOP$.

Composición paralela

La composición paralela $P \underset{A}{\|} Q$ de P y Q, sobre el conjunto de eventos A ($A = \alpha(A) \cap \alpha(A)$) fuerza a P y Q a involucrarse y sincronizarse en todo evento $a \in A$, teniendo un comportamiento independiente uno del otro con respecto a todos eventos $e \notin A$. Así, si dos procesos comparten un evento, ambos deben estar dispuestos a consumirlo al mismo tiempo; en caso contrario, uno de ellos deberá esperar a que el otro proceso esté en condiciones de hacerlo, es decir, los eventos no se pierden, sino que los procesos se bloquean. Las leyes algebraicas

que describen esta semántica son las siguientes:

$$e, f \in \alpha P \cap \alpha Q \wedge e \neq f \Rightarrow f \rightarrow P||e \rightarrow Q = STOP \ \ (Interbloqueo)$$

$$e \rightarrow P||e \rightarrow Q = e \rightarrow (P||Q) \ \ (Sincronizacion)$$

$$e \in \alpha P \wedge f \notin \alpha Q \Rightarrow f \rightarrow P||e \rightarrow Q = f \rightarrow (P||e \rightarrow Q) \ \ (IEOF)$$

En el caso del *interbloqueo*, siendo e y f eventos compartidos, cada uno de los dos procesos está esperando a que el otro esté dispuesto a comunicar el evento que necesita para progresar, pero como ambos están en la misma situación de espera, el progreso es imposible.

La segunda ley, *sincronización*, muestra otro aspecto del modelo sincrónico adoptado por CSP: si dos procesos deben interactuar a través de un evento común, entonces ambos consumen el evento al mismo tiempo.

La tercera ley, llamada *IEOF* (*independent events occur first*), o sea, los eventos independientes se consumen primero, representa la tercera posibilidad: los eventos que no son compartidos tienen prioridad sobre los eventos comunes; en caso contrario, la composición paralela no podría progresar, ya que para consumir el evento común ambos procesos deben estar en condiciones de hacerlo, y uno de ellos no lo está.

Entrelazamiento

$P|||Q$ corresponde a una composición paralela sobre un conjunto de eventos vacío, es decir, cada proceso se comporta de manera independiente del otro con ninguna sincronización en ningún evento. De ello,

$$P|||Q = P \underset{\emptyset}{\|} Q$$

Ocultamiento

El proceso $P \setminus A$, con $A \subseteq \alpha P$ es el proceso que se comporta como P; sin embargo todas las comunicaciones en el conjunto A se hacen internas al proceso, o sea, son invisibles al exterior. De esta manera, se extraen del control del entorno. Por ejemplo, si $P = a \rightarrow b \rightarrow STOP$, entonces $P \setminus b = a \rightarrow STOP$.

La suposición del *progreso máximo*[1] dicta que no puede pasar ningún tiempo mientras se ofrecen los eventos ocultos, en otras palabras, los eventos ocultos ocurren en cuanto estén disponibles.

Presentamos algunas leyes básicas de ocultación:

$$(P||Q) \setminus X = P \setminus X||Q \setminus X, \;\; si \;\; \alpha P \cap \alpha Q \cap X = \emptyset$$

$$P \sqcap Q) \setminus X = P \setminus X \sqcap Q \setminus X$$

$$a \rightarrow P \setminus X = \begin{cases} P \setminus X & a \in X \\ a \rightarrow (P \setminus X)\backslash & a \notin X \end{cases}$$

Renombramiento

El renombramiento es un mecanismo para renombrar procesos cambiando el nombre de sus eventos. Para ello, primero, se define una función que mapea cada nombre de evento que debe ser renombrado a otro nombre, y luego se aplica la función a todo el proceso.

El proceso de renombrado $f(P)$ deriva sus comportamientos del comportamiento del proceso P en eso, si P puede realizar un evento a, el proceso $f(P)$ pueda realizar $f(a)$.

Recursión

La recursión $\mu X.P$ representa un término sintáctico que podemos considerar como la solución a la equación $X = P$. La variable X está ligada mediante el operador μ

[1] Un evento de sincronización ocurre lo más pronto que los participantes estén preparados en consumirlo.

3.2.3. Semántica de modelado

El modelo semántico del álgebra de procesos se puede dar en diferentes formas, por consiguiente, los enumeramos a continuación y representamos sus principales características:

- Semántica denotacional: la semántica denotacional provee significado a los programas de computadora en forma de funciones matemáticas que transforman las entradas en salidas del programa. Una característica de esta aproximación digna de mencionar es la *composicionalidad*, gracias a la cual el significado de un programa se define en términos del significado respectivo de sus subcomponentes.

- Semántica operacional: típicamente, el significado de los términos de procesos CSP se explican en términos de máquinas de estados infinitos, a saber, los *sistemas de transición etiquetadas*, con nodos representando estados, y aristas representado acciones. Es la forma más cercana a una implementación, ya que representa los programas como secuencias de *pasos de ejecución* y figura de forma perceptible en los algoritmos de comprobación de modelos *Model checking*. Esta semántica viene, en la mayoría de las veces, dotada de la noción de *bisimulación*, que provee un mecanismo para determinar la equivalencia entre procesos.

- Semántica algebraica: se puede considerar como la forma más abstracta de formalizar un lenguaje. Este modelo semántico intenta capturar el significado de un programa a través de la definición de una serie de leyes algebraícas: se axiomatizan ciertas propiedades y de esta axiomatización se deducen teoremas sobre los constructores del lenguaje de programación. Un ejemplos de estos algebraicas leyes puede ser la ley explicada anteriormente: $STOP; P = STOP$

Notaciones

Antes de dar la semántica de CSP, definimos algunas notaciones que van a ser útiles para describir sus modelos:

Las secuencias van a figurar de manera prominente en el modelo denotacional de CSP, por ello necesitamos introducir ciertas definiciones respecto de las mismas.

- $\sigma = \langle \sigma_1, \sigma_2, \sigma_3, ..., \sigma_n \rangle$, denota una secuencia σ de longitud n.
- $\langle\,\rangle$, representa una secuencia vacía.
- $\varphi(\sigma)$, denota el conjunto de elementos que aparecen en σ
- $\sharp\ \sigma$, devuelve el número de elementos en una secuencia σ, y toma el valor ∞ si la secuencia es infinita.
- $head(\sigma)$, representa el primer elemento en una secuencia σ, por ejemplo: $head(\langle a,\, b,\, c \rangle) = \langle a \rangle$.
- $\sigma^\frown s$, representa la concatenación de las dos secuencias σ y s.

  Sea, $\sigma = \langle \sigma_1, \sigma_2, ...\sigma_n \rangle$ y $s = \langle s_1, s_2, ...s_n \rangle$, $s^\frown \sigma = \langle s_1, s_2, ...s_n, \sigma_1, \sigma_2, ...\sigma_n \rangle$
- $s \leq \sigma$, denota una relación de orden parcial[2] entre s y σ, definida como:

  $s \leq \sigma$ si $\exists \delta_1, \delta_2$ tal que: $\sigma = \delta_1^\frown s^\frown \delta_2$

  - si $\delta_1 = \langle\,\rangle$ entonces $\sigma = s^\frown \delta_2$ decimos que s es el prefijo de σ
  - si $\delta_1 = \langle\,\rangle$ y $\delta_2 = \langle\,\rangle$ entonces $\sigma = s$
- $s \upharpoonright A$, denota la restricción de la secuencia de elementos s a un conjunto de eventos A, esto es, la secuencia de elementos de s que están en A.

  Por ejemplo: $\langle b,\, c,\, d,\, a \rangle \upharpoonright \{a, c\} = \langle c,\, a \rangle$.

  Deducimos:

$$\langle\ \rangle \upharpoonright A = \langle\ \rangle$$

$$(\langle a \rangle^\frown t) \upharpoonright a = \langle a \rangle^\frown (t \upharpoonright A)\ \ si\ \ a \in A$$

$$(\langle a \rangle^\frown t) \upharpoonright A = (t \upharpoonright A)\ \ si\ \ a \notin A$$

[2]Un orden parcial es una relación binaria sobre un conjunto X que es reflexiva, antisimétrica, y transitiva

- $\sigma \setminus A$, representa una subsecuencia de σ, cuyos elementos no pertenecen a A. Por ejemplo, $\langle a, b, c, a\rangle \setminus \{a, b\} = \langle c\rangle$

- Sea A un conjunto arbitrario, denotamos con A^* el conjunto de secuencias finitas de elementos del conjunto A, con $\mathbb{P}A$ denotamos los subconjuntos de A, y con $A^{\surd}$ el conjunto que incluye los elementos de A y el evento de terminación $\surd$.

$$A^* = \{\sigma | \sharp\sigma < \infty \wedge \forall a, a \leq \sigma \Rightarrow a \in A\}$$

$$\mathbb{P}A = \{X | X \subseteq A\}$$

$$A^{\surd} = A \cup \{\surd\}$$

3.2.4. Semántica de trazas

La manera más simple de entender el comportamiento de un proceso CSP es observar su ejecución y registrar las trazas o secuencias de eventos que consume[3], en el orden en que éstos van ocurriendo. En general, las trazas pueden ser finitas o infinitas, tanto debido a que se termina la observación del proceso como a que el proceso no se involucra en más comunicaciones con el entorno; o infinitas, porque se observa al proceso infinitamente, y éste se involucra en infinitas comunicaciones. Nos limitaremos a registrar únicamente las trazas finitas debido a que éstas son suficientes para describir la mayoría de los procesos [RHB97].

Para cada proceso P, la $tr(P)$, es el conjunto de todas las trazas finitas de P. $tr(P)$, satisface las propiedades siguientes:

- $tr(P) \neq \emptyset$, contiene siempre la traza vacía $\langle\rangle$.

- $tr(P)$, es *cerrado a prefijos*: si $s^\frown\sigma \in tr(P)$, entonces $s \in tr(P)$.

El conjunto de todos los conjuntos no vacíos, cerrados a prefijos se denomina ***modelo de trazas***, etso es, el conjunto de todas las posibles representaciones de un proceso respecto de sus trazas. Denotamos el modelo de trazas con M_T

[3]Los eventos internos no se registran en una traza temporizada, dado que no son visibles al interfaz, y su ocurrencia no puede tener ninguna influencia directa en ningún otro proceso.

y definimos una función semántica $F_T : CSP \to M_T$, que mapea cada proceso CSP con su modelo de trazas. Aquí sólo daremos la semántica de trazas de algunos procesos. Consulten [RHB97] para más detalles.

$$F_T(STOP) = \{\langle\,\rangle\} \tag{3.2.1}$$

$$F_T(SKIP) = \{\langle\,\rangle, \langle\surd\rangle\} \tag{3.2.2}$$

$$F_T(a \to P) = \{\langle\rangle\} \cup \{\langle a\rangle^\frown tr \mid tr \in F_T(P)\} \tag{3.2.3}$$

$$F_T(P \Box Q) = F_T(P) \cup F_T(Q) \tag{3.2.4}$$

$$F_T(P \sqcap Q) = F_T(P) \cup F_T(Q) \tag{3.2.5}$$

$$F_T(P||Q) = \{tr|\varphi(tr) \in (\alpha P \cup \alpha Q)^* \wedge tr \upharpoonright \alpha P \in F_T(P) \tag{3.2.6}$$

$$\wedge\, tr \upharpoonright \alpha Q \in F_T(Q)\}$$

$$F_T(P;Q) = \{tr \mid tr \in F_T(P) \wedge \surd \notin \varphi(tr)\} \tag{3.2.7}$$

$$\cup$$

$$\{tr_1^\frown tr_2 \mid tr_1^\frown \langle\surd\rangle \in F_T(P) \wedge tr_2 \in F_T(Q)\}$$

$$F_T(f(P)) = \{f(tr) \mid tr \in F_T(P)\} \tag{3.2.8}$$

1. El proceso $STOP$ no realiza ningún evento, de ello, su ejecución sólo puede realizar la traza vacía.

2. El proceso $SKIP$, sólo puede realizar el evento de terminación exitosa $\surd$ si el entorno lo ofrece.,o bien, sólo la traza vacía si el entorno no llega a ofrecer $\surd$.

3. Si el entorno ofrece el evento A, el proceso $a \to P$ va a realizar el evento a, y pasa a comportarse como $F_T(P)$, sino, el proceso no puede progresar y su comportamiento va ser una traza vacía.

4. la elección externa entre P y Q se resuelve a favor de uno de los dos procesos P o Q, depende del primer evento visible elegido por el entorno. Asimismo, El proceso $P \Box Q$ se comporta como P, o como, Q. De ello su comportamiento en términos de trazas se define con la unión de las trazas de P y las trazas de Q [4].

5. El proceso $P \sqcap Q$ se comporta bien como P, o bien como, Q, la elección se resuelve de manera interna en el proceso sin que el entorno tenga ningún control sobre ella. Asimismo, las posibles ejecuciones del proceso $P \sqcap Q$, son las posibles ejecuciones de P o las posibles ejecuciones de Q.

6. El proceso $P||Q$ puede realizar los eventos en los alfabetos de comunicación de P y de Q, de ello, su traza está formada por eventos en $\alpha P \cup \alpha Q$. y las trazas del mismo proyectado al alfabeto de comunicación de P, pertenecen a las trazas de P, y las trazas del mismo proyectado al alfabeto de comunicación de Q, pertenecen a las trazas de Q,

7. El proceso $P; Q$ se comporta como P, hasta que P termine su ejecución de manera exitosa [5](acepta el evento $(\surd)$)), y pasa el control al proceso Q. Si el proceso P no termina de manera exitosa el proceso $P; Q$ no pasa a comportarse como $F_T(Q)$.

8. El comportamiento del proceso $f(P)$ es *indistinguible* (bisimilar) del de P si se renombran los eventos de su traza.

- Anotamos con $P \setminus \sigma$, el comportamiento de P después de que se completa la traza σ.

  $F_T(P \setminus \sigma) = \{tr \mid \sigma^\frown tr \in F_T(P)\}$

$F_T(P)$ tiene dos utilidades: permite dar significado y semántica a la notación de CSP, y también especificar el comportamiento requerido de un proceso, es decir, las propiedades deseadas para éste.

[4]Las trazas de **las alternativas etiquetadas** se pueden calcular con simplemente reescribirlas como elecciones externas (esto es, reemplazando todas las construcciones | con las construcciones $\Box$).

[5]La terminación exitosa de P, se hace internamente en este, y no implica la terminación del proceso $P; Q$

3.2.4.1. Acerca de la semántica de $\Box$ y $\sqcap$

Las ecuaciones 3.2.5 y 3.2.4, muestran que las trazas de la elección interna y la elección externa son las mismas. De ahí, deberíamos concluir que $P \Box Q = P \sqcap Q$, cosa que no puede ser verdad. Los dos procesos no pueden ser iguales, dado que uno implica determinismo y el otro no. Esto lleva a pensar que la semántica de trazas no provee una descripción completa de P y que no permite hacer una distinción entre las dos elecciones en cuestión.

Las trazas representan lo que un proceso hace en una secuencia del mismo, pero no se interesa por lo que debe hacer o puede rechazar. Por ejemplo, la traza de los dos procesos :

$$(a \rightarrow STOP \ \Box \ b \rightarrow STOP) \text{ y } (a \rightarrow STOP \ \sqcap \ b \rightarrow STOP)$$

son iguales; sin embargo, el primer proceso debe responder siempre al entorno cuando uno o los dos eventos a y/o b estén disponibles, a diferencia del segundo, que puede no aceptar ninguno de los dos eventos, siempre que el entorno no le ofrezca ambos al mismo tiempo [6]. Por lo tanto, si a las trazas de los dos procesos sumamos sus *rechazos*, podremos distinguirlos. En general, una manera de distinguir los procesos y completar la semántica de CSP, además del modelo de trazas consiste en, considerar los rechazos de los procesos.

3.2.5. Semántica de rechazos

Los rechazos de P es un conjunto de eventos [7] que el proceso declina, sin importar durante cuánto tiempo se los ofrece el entorno. Para un proceso P, los $rechazos(P)$ satisfacen siempre las propiedades siguientes:

- $rechazo(P) \neq \emptyset$: contiene siempre la traza vacía $\langle\rangle$.
- $rechazo(P)$ es *cerrado para subconjuntos*:es decir, si$X \in rechazos(P)$ y $Z \subseteq Z$ entonces $Z \in rechazos(P)$.

El conjunto de todos los conjuntos no vacíos, cerrados para subconjuntos, se denomina *modelos de rechazos*, esto es, el conjunto de conjuntos de rechazos

[6] pero incluso si el entorno le ofrece ambos eventos, el proceso puede rechazar implicarse en uno de ellos de manera autónoma e imprevisible para sus observadores.

[7] pertenecientes al alfabeto de comunicación de P, αP.

iniciales de P, es decir, los conjuntos de eventos que el proceso rechaza cuando sólo se ha producido la traza vacía.

Denotamos el modelo de rechazos con M_R. Y definimos una función semántica $F_R : CSP \to M_R$, que representa los procesos CSP a sus correspondientes modelos de rechazos. Aquí, sólo daremos la semántica de rechazos de algunos procesos, para más detalle se puede consultar [RHB97].

$$F_R(STOP) = \mathbb{P}A \tag{3.2.9}$$

$$F_R(SKIP_A) = \mathbb{P}A \tag{3.2.10}$$

$$F_R(a \to P) = \{X \mid X \subseteq \alpha P \setminus \{a\}\} \tag{3.2.11}$$

$$F_R(P \Box Q) = F_R(P) \cap F_R(Q) \tag{3.2.12}$$

$$F_R(P \sqcap Q) = F_R(P) \cup F_R(Q) \tag{3.2.13}$$

$$F_R(P||Q) = \{X \cup Y \mid X \in F_R(P) \wedge Y \in F_R(Q)\} \tag{3.2.14}$$

$$F_R(F(P)) = \{F(X) | X \in F_R(P)\} \tag{3.2.15}$$

1. El proceso $STOP$, rechaza cualquier evento que le puede ofrecer el sistema.

2. El proceso $SKIP$, rechaza cualquier evento que le puede ofrecer el sistema, a parte del evento $\surd$[8].

3. El proceso $a \to P$, rechaza cualquier conjunto de eventos que no contiene el evento a.

4. Si X no es un rechazo de P, entonces P no puede rechazar X y así, tampoco lo puede rechazar $P \Box Q$. Del mismo modo, si X no es un rechazo de Q, entonces, no es un rechazo de $P \Box Q$. Asimismo, si ambos P y Q puede rechazar X, lo mismo haría P y Q.

[8] $\surd \notin \mathbb{P}A$, sino se anotaría $\mathbb{P}A^{\surd}$.

5. si P puede rechazar X, también lo puede rechazar $P \sqcap Q$, si P es el proceso que ha sido seleccionado. De la misma manera, cualquier rechazo de Q es también un rechazo de $P \sqcap Q$. Entonces, los rechazos de $P \sqcap Q$ son la unión de los rechazos de los ambos proceso P y Q.

6. el proceso $P||Q$, puede rechazar los eventos rechazados por P y también los eventos rechazados por Q, es decir, el proceso puede rechazar la unión de los rechazos de P y los rechazos de Q.

7. Si P rechaza un elemento a, $f(P)$ rechaza $f(b)$, asimismo, los rechazos de $f(P)$ son $f(X)$, tal que X es un rechazo de P.

La introducción del concepto de rechazos permite establecer una clara distinción entre los procesos deterministas y no-deterministas.

Se dice que un proceso es determinista si nunca puede rechazar ningún evento que pertenezca a su alfabeto de comunicación y que ocurra en una de sus posibles trazas. Formalmente:

$$P \text{ es determinista} \equiv \forall s \in F_T(P), \forall X \in F_R(P)) \equiv X \cap (\varphi(s)) = \emptyset$$

Por ello, un proceso no determinista es un proceso que no cuenta con esta propiedad, esto es, rechaza eventos que le ofrece el entorno, y que están en su alfabeto de comunicación, por lo cual el entorno entiende que el proceso debe aceptar estos eventos, y al declinar la comunicación se considera no-determinista.

3.2.6. Semántica de fallos

Necesitamos saber, no solamente lo que un proceso P puede rechazar después de una traza vacía, sino también, lo que puede rechazar después de cualquiera de sus trazas. Los fallos representan una combinación de las trazas y rechazos de procesos. Formalmente, los fallos de un proceso P, es el conjunto $F(P)$, tal que:

$$F_F(P) = \{(tr, X) | tr \in F_T(P) \wedge X \in F_R(P/tr)\}$$

Sea A un conjunto de eventos temporizados.

$$F_F(STOP) = \{(\langle\rangle, X) | X \subseteq A^{\surd}\} \tag{3.2.16}$$

$$F_F(SKIP) = \{(\langle\rangle) \mid X \subseteq A\} \cup \{(\langle\surd\rangle, X) | X \subseteq A^{\surd}\} \tag{3.2.17}$$

$$F_F(a \rightarrow P) = \{(\langle\,\rangle, X) \mid a \notin X\} \tag{3.2.18}$$

$$\cup$$

$$\{(\langle a\rangle \frown tr, X) \mid (tr, X) \in F_F(P)\}$$

$$F_F(P \sqcap Q) = F_F(P) \cup F_F(Q) \tag{3.2.19}$$

$$F_F(P \,\Box\, Q) = \{(\langle\rangle, X) \mid X \in F_R(P) \cap F_R(Q)\} \tag{3.2.20}$$

$$\cup$$

$$\{(tr, X) | tr \neq \langle\rangle \wedge (tr, X) \in F_F(P) \cup F_F(Q)\}$$

$$F_F(P||Q) = \{(t, X \cup Y) | X \setminus (\alpha P \cap \alpha Q)^{\surd} = Y \setminus (\alpha P \cap \alpha Q) \tag{3.2.21}$$

$$\wedge\, (t \uparrow \alpha P, X) \in F_F(P)$$

$$\wedge\, (tr \uparrow \alpha P, Y) \in F_F(Q)\}$$

$$F_F(f(P)) = \{f(X) | X \in F_F(P)\} \tag{3.2.22}$$

$$F_F(P; Q) = \{(tr, X) \mid \langle\surd\rangle \notin tr \tag{3.2.23}$$

$$\wedge\, (tr, X^{\surd}) \in F_F(P)$$

$$\cup \{(s \frown tr, X) \mid s \frown \langle\surd\rangle \in F_T(P) \wedge (tr, X) \in F_F(Q)\}$$

1. El proceso $STOP$, no realiza ningún evento y rechaza cualquier evento que se le comunica.

2. El proceso $SKIP$, solo puede aceptar el evento $\surd$ si el entorno lo comunica.

3.3. CSP+T

CSP+T [Ž94] es un lenguaje de especificación de tiempo real,que proporciona perspectivas muy valiosas en la especificación de eventos temporales complejos.

Dentro del grupo de lenguajes derivados de CSP, con el objetivo de poder describir intervalos temporales, cabe mencionar Timed CSP y CSP+T, siendo el segundo una aproximación más simple que la primera, por lo que proporciona un menor poder descriptivo aunque, lo suficientemente completo para describir formalmente el comportamiento de un conjunto de procesos deterministas con restricciones temporales. CSP+T [Zic91] es un lenguaje de especificación formal adecuado para la mayoría de los sistemas de tiempo real.

3.3.1. Modelo de tiempo

Una de las primeras decisiones que hay que tomar a la hora de modelar un sistema temporizado se refiere a la manera de modelar el tiempo. El tiempo se puede modelar como continuo o discreto [Sch89] [RR88].

Para la captura del tiempo de respuesta a estímulos con una buena precisión se suele usar el modelo de tiempo continuo, que supone que entre dos instantes de tiempo, t_1 y t_2, siempre existe otro instante de tiempo t . No obstante, en sistemas de computación el tiempo se mide con relojes.

Un reloj divide un intervalo continuo de tiempo a una secuencia finita de instantes, con lo cual cada instante corresponde a un *tick* de reloj. En efecto, entre dos *ticks* seguidos no se puede capturar ningún instante de tiempo; así, el intervalo entre ellos se va a ignorar por el sistema. Por esta razón, entre dos instantes seguidos medidos por un reloj, no existe otro instante de tiempo medido por el mismo reloj [Joe00].

Anotamos con $\Re^+$ el conjunto de los números reales positivos, y con $\mathcal{Z}$ el conjunto de lo números enteros no negativos.

$\forall t \in \Re^+, \exists t_z \in Z \; y \; \epsilon \in [0,1) \; tal \; que \; t_z \leq t \leq t_z + \epsilon$ Definimos une constante $\phi \in [0,1)$. El valor del número real t, cambia al valor del número entero, precedente o siguiente, tal y como define la función Φ:

$$\Phi(t) = \begin{cases} t_z & if \epsilon < \phi \\ t_z + 1 & if \epsilon \geq \phi \end{cases}$$

Esta aproximación permite transformar un intervalo de tiempo continuo a otro intervalo de tiempo discreto; en el sentido de que en un intervalo de tiempo continuo consta de una secuencia finita de instantes de tiempo. De esta manera, la precisión temporal de un sistema depende de las unidades de medida que se utilizan en el sistema. Como *dominio de tiempo* elegimos el conjunto de

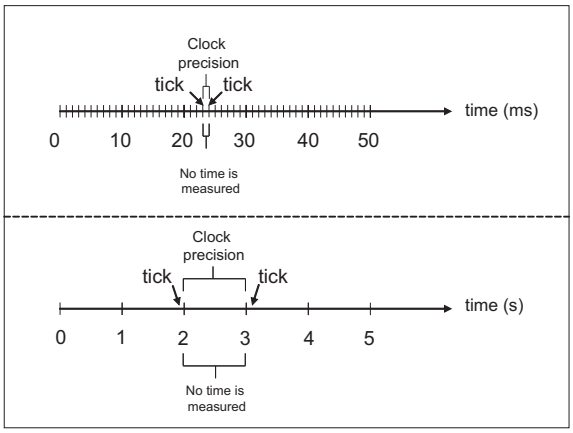


Figura 3.1: Reloj local

los números positivos $\Re^+$. La secuencia de tiempo $\tau = \tau_1, \tau_2, \tau_3 \ldots$, es una secuencia infinita de valores de tiempo, $\tau_i \in \Re^+$ con $\tau_i > 0$, que satisface las propiedades siguientes.

1. ***monotonía:*** τ crece de manera estrictamente monótona, es decir, $\tau_i < \tau_{i+1}$ para cada $i \geq 1$
2. ***progreso:*** para cada $\tau \in \Re$, existe $i \geq 1$ tal que $\tau_i > \tau$
3. ***tiempo de Newton:*** el tiempo progresa en todos los procesos con la misma velocidad, y con el mismo y único marco de *tiempo global*.

3.3.2. Eventos temporizados

Los eventos temporizados se definen con un par $(\Sigma, \Im^+)$, con

- Σ: es el conjunto de eventos,
- $\Im$: es el conjunto de tiempos de ejecuciones de los eventos $e \in \Sigma, (\Im \subset \Re^+)$

Definimos las funciones:

- $lot : \sigma \rightarrow R$, es la función que hace corresponder cada evento $e \in \sigma$ a un limite inferior de tiempo de ocurrencia del evento e
- $hot : \sigma \rightarrow \Re^+$, es la función que hace corresponder cada evento $e \in \sigma$ a un limite superior de tiempo de ocurrencia del evento e, tal que $\forall e \in \Sigma \;\; lot(e) \leq hot(e)$.
- $T : \Sigma \rightarrow \Re^+$, es la función que hace corresponder cada evento con su tiempo de ejecución.

Cualquier evento debe ocurrir en un tiempo, que cumple con sus restricciones temporales, para ello:

$$\forall e \in \Sigma \quad T(e) \in [lot(e),\, hot(e)\,]$$

Introducimos la noción de:

1. ***evento marcador***, es un evento temporizado me, cuyo tiempo de ejecución es registrado en la varaible marcadora v, con $T(me) = v$ y $v \in \Re^+$.
2. ***evento restringido***, es un evento temporizado re, cuyo tiempo de ejecución depende de tiempo de ejecución de un evento marcador me que lo preceda. Así, el tiempo de ocurrencia de re está restringido a un intervalo de tiempo (llamado, *intervalo de habilitación*), donde el limite inferior representa un instante de tiempo relativo a la variable marcadora v. Formalmente,

$$T(re) \in [T(me), T(me) + \tau] \;\; , con \;\; \tau \in \Re^+$$

3.3.3. Intervalos de habilitación

Los intervalos de habilitación de acuerdo al artículo de Zic [Ž94] se pueden definir de las siguientes tres formas equivalentes:

- $P' = 0 \rightarrow a \bowtie v \rightarrow [3,5].c \rightarrow STOP$
- $P' = 0. \rightarrow a \bowtie v \rightarrow [rel(3,v), rel(5,v)].c \rightarrow STOP$
- $P' = 0. \rightarrow a \bowtie v \rightarrow E.c \rightarrow STOP donde E = \{t | rel(3,v) \leq t \leq rel(5,v)\}$

En cualquiera de los casos, los intervalos de habilitación se definen utilizando instantes de tiempo relativos, mientras que las variables marcadoras registran el tiempo de forma absoluta.

De las tres definiciones anteriores, nos quedamos con la primera forma, que es la más simple y compacta.

Ahora bien, también hemos definido el intervalo de habilitación de otra forma más compacta, para los casos en los que no hay que indicar un intervalo de tiempo relativo (límite inferior y límite superior) a partir del momento en el que se registra el instante de tiempo utilizando una variable marcadora, esto es :

$$I(intervalo, v) = [rel(0, v), rel(intervalo, v)]$$

De acuerdo a esta nueva definición, los siguientes dos términos de procesos son equivalentes:

$P' = 0.\star \rightarrow a \bowtie v \rightarrow [0, 2].c \rightarrow STOP$

$P' = 0.\star \rightarrow a \bowtie v \rightarrow I(2, v).c \rightarrow STOP$

La diferencia esencial entre las dos definiciones es que el primer argumento de la función I indica la duración, 2 unidades de tiempo, y el segundo argumento indica la variable marcadora que se toma como referencia. El resto, c, es el evento que tiene que ocurrir dentro del plazo de tiempo determinado por el intervalo de habilitación y en el que se implica el proceso P'.

Para definir un retardo (*delay*) teniendo en cuenta lo anterior podríamos utilizar la notación anterior, pero impidiendo que se active ningún evento dentro del plazo especificado por el intervalo de habilitación. Por ejemplo,

$$P = 1.\star \rightarrow a \bowtie v \rightarrow I(3, v) \rightarrow c \rightarrow STOP$$

En el ejemplo anterior, por tanto:

- Se establece un origen de tiempo en el instante 1 (unidad de tiempo).
- Cuando ocurre el evento a, se marca el instante de tiempo en la variable marcadora v ($v \in]1, +\infty[$).
- A continuación, se establece un retardo de 3 unidades de tiempo después de que ocurra el evento a, registrado en la variable marcadora v.
- Después de que haya transcurrido el retardo, se espera a que ocurra el evento c.

- Finalmente, para la ejecución del proceso.

3.3.4. Operadores de CSP+T

La sintaxis de CSP+T es un superconjunto de la sintaxis de CSP. Anotamos con Σ cualquier conjunto finito de eventos. En la notación que sigue tenemos $a \in \Sigma,\ B \subseteq \Sigma$. Los términos de CSP+T se construyen conforme a la gramática siguiente:

$$\begin{aligned} P ::= \quad & STOP \mid SKIP \mid 0.\star \rightarrow P \mid a \bowtie v \rightarrow P \mid I(T,t).a \rightarrow P \\ & \mid I(T,t) \rightarrow P \mid P \sqcap Q \mid P \Box Q \mid P; Q \mid P \setminus A \mid P[B]Q \\ & \mid P|||Q \mid f(P) \mid \mu X.P \end{aligned}$$

Las diferencias principales incluidas en la sintaxis de CSP+T con respecto a la de CSP son:

Prefijado de eventos

- $0.\star \rightarrow P$ representa el proceso prefijado por el *evento de instanciación* $\star$. Este evento es único en el sistema y representa el tiempo global en lo cual los procesos del sistema se pueden activar.

- $a \bowtie v \rightarrow P$ es el proceso que primero se involucra en el evento a, y luego se comporta exactamente como describe P. *El operador de captura de tiempo* ($\bowtie$) es asociado a la función de sello de tiempo $v = T(a)$, que permite registrar en el tiempo de ocurrencia de un evento a en la variable marcadora v.

- En $I(T,t).a \rightarrow P$, el proceso P puede aceptar el evento a, solamente, si se ofrece dentro del intervalo de tiempo $[t, t+T]$.

- un proceso P puede estar prefijado por un intervalo de tiempo, $I(T,t) \rightarrow P$ es el proceso que se comporta como P, inmediatamente, cuando se agota el intervalo de tiempo I. Se puede considerar este intervalo como *un evento especial de retardo.*

Composición secuencial

$P;Q$, representa la composición secuencial de P y Q. El proceso $P;Q$ se comporta como P, hasta que P, elije terminar[9] y pasa a comportarse como Q. El proceso $P;Q$ puede bloquearse si P, termina a un instante t mayor al limite superior del *intervalo de habilitación* asignado al primer evento del proceso Q.

Por ejemplo, consideramos $P = a \bowtie v \rightarrow b$, y $Q = E.c \rightarrow STOP$, con $E = I(4, v+2)$.

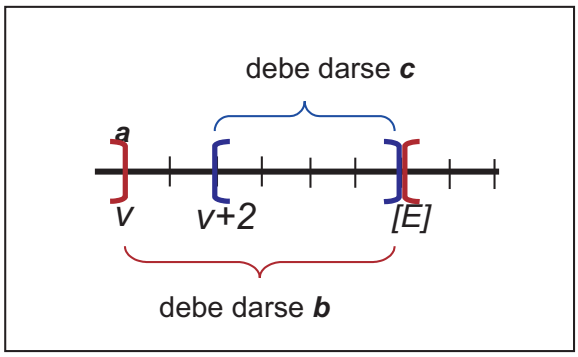


Figura 3.2: Restricciones de tiempo en la ejecución de los eventos b y c

El evento c, debe ocurrir en 2 a 6 unidades de tiempo a partir del tiempo de ocurrencia del evento a. El tiempo de ocurrencia del evento b debe ser inferior estrictamente al límite superior del intervalo de habilitación del evento c. En caso contrario, el proceso para su ejecución (pasa a comportarse como el proceso $STOP$) inmediatamente después de realizar el evento b. Sea $[E]$, el límite superior del intervalo E. La composición secuencia se debe rescribir como sigue:

$$P;Q = a \bowtie v \rightarrow [v, t[.b \rightarrow E.c \rightarrow STOP \ \square \ ([E], \infty).b \rightarrow STOP$$

Composición paralela

$P \underset{B}{\|} Q$ representa la composición paralela de P y Q, sobre un interfaz B. Si el alfabeto B es parcialmente disjunto con respeto al alfabeto de comunicación de P y Q. La sincronización de los dos procesos P y Q depende de:

[9] P se involucra en el evento de terminación exitosa $\surd$

1. el conjunto de eventos en la intersección de αP y αQ

2. La intersección de los intervalos de habilitación de los eventos en B.

Por ejemplo, consideramos los dos procesos simples $P = E_1.a \rightarrow P$ y $Q = E_2.a \rightarrow Q$ con $\alpha P = \alpha Q$. La semántica de la composición paralela de los dos proceso depende de si los valores tomados por sus intervalos E_1 y E_2, son idénticos, entrelazan parcialmente o disjuntos:

- si $E_1 = E_2$, el proceso $P \underset{B}{\|} Q$ se involucra en el evento de sincronización a a cualquier instante dentro del intervalo E_1.

- si $E_1 \neq E_2$ *y* $E_1 \cap E_2 \neq \emptyset$, el proceso $P \underset{B}{\|} Q$ sólo puede involucrarse en el evento de sincronización a en tiempo $t \in E_1 \cap E_2$.

- si $E_1 \neq E_2$ y $E_1 \cap E_2 = \emptyset$, ninguna sincronización en el evento a es posible, así, el proceso $P \underset{B}{\|} Q$ se bloquea.

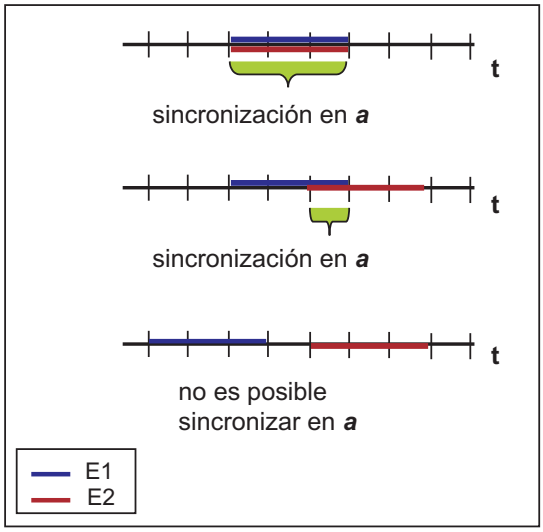


Figura 3.3: Sinronización en el evento A

3.4. Semántica denotacional para CSP+T

3.4.1. Notaciones

Antes de establecer los modelos semánticos de CSP+T, primero definimos algunas notaciones que nos van a ser útiles: sea $a \in \Sigma^{\surd}$ y $t \in \Re^{+} \cup \infty$

- $\sigma = \langle(\sigma_1, t_1), (\sigma_2, t_2) \ldots (\sigma_n, t_n), \rangle$, representa una secuencia de eventos temporizados(σ_i, t_i)
- $\varphi(s) = \{(a, t) | s \upharpoonright a \neq \emptyset\}$
- $begin(\sigma) = inf\{t \,|\, (a, t) \in \varphi(\sigma)\}$
- $end(\sigma) = sup\{t \,|\, (a, t) \in \sigma(s)\}$
- $begin(\langle\rangle) = end(\langle\rangle)$
- $head(\langle(t,\, a)^\frown s\rangle) = a$
- $s \uparrow [t_1, t_2]$, representa la proyección de la traza temporizada s sobre un intervalo de tiempo $[t_1, t_2]$.
- $s + t$, incrementa con t el tiempo de ocurrencia de todos los eventos en la secuencia s, por ejemplo: $\langle(a, 1), (b, 3)\rangle + 2 = (\langle a, 3), (b, 5)\rangle$.
- s_1 **in** $s_2 \Leftrightarrow \exists s_3 \in ttr$ *tal que* $s_1 = s_2 \frown s_3$
- s_1 **notin** $s_2 \Leftrightarrow \nexists\, s_3 \in ttr$ *tal que* $s_1 = s_2 \frown s_3$

3.4.2. Modelo de trazas temporizadas

Un sistema de tiempo real se puede definir como un conjunto de trazas temporizadas, que representa el comportamiento potencial del sistema como secuencia de eventos temporizados [10] que se pueden observar durante la ejecución de un sistema. Por cada proceso P, las $TTrazas(P)$ verifica las propiedades siguientes:

[10] Con un tiempo no decreciente, y los eventos se registran en el orden temporal

- $Traza(P) \neq \emptyset$
- *cerrado para-prefijos*
- Ordenado en tiempo:

$$s \in TTrazas(P) | \exists t_1, t_2 : \mathrm{Re}^+; a_1, a_2 : \Sigma^{\surd} \langle (t_1, a_1), (t_2, a_2) \rangle \leq s \Rightarrow t_1 \leq t_2.$$

Denominamos al conjunto de todas las posibles representaciones de un proceso en trazas temporizadas, un *modelo de trazas temporizadas*, que anotamos con M_{TT}. Definimos la función semántica $F_{TT} : CSP + T \rightarrow M_{TT}$, que representa cada proceso CSP+T con un modelo en M_{TT}.

$$
\begin{aligned}
F_{TT}(t_0.\star \rightarrow P) &= \{\langle\rangle\} \cup \{(\star, t_0)^\frown X \mid X \in F_{TT}(P)\} && (3.4.1)\\
F_{TT}(e \bowtie v \rightarrow P) &= \{\langle\rangle\} \cup \{(e,\, v)^\frown X \mid X \in F_{TT}(P)\} && (3.4.2)\\
F_{TT}(I(T,t).e \rightarrow P) &= \{\langle\rangle\} && (3.4.3)\\
&\quad \cup \\
&\quad \{(e,\, t_e)^\frown X \mid X \in F_{TT}(P) \wedge t_e \in I(T,t)\} \\
F_{TT}(I(T,t) \rightarrow P) &= \{\langle\rangle\} \cup \{(\surd,\, T+t)^\frown X \mid X \in F_{TT}(P)\} && (3.4.4)\\
F_{TT}(P \ \Box \ Q) &= F_{TT}(P) \ \cup \ F_{TT}(Q) && (3.4.5)\\
F_{TT}(P \ \sqcap \ Q) &= F_{TT}(P) \ \cup \ F_{TT}(Q) && (3.4.6)\\
F_{TT}(P; Q) &= F_{TT}(P)^\frown F_{TT}(Q) && (3.4.7)\\
F_{TT}(P \underset{B}{\|} Q) &= \{\langle\rangle\} \cup \{tr \in M_{TT} \mid tr \upharpoonright \alpha P \in F_{TT}(P) && (3.4.8)\\
&\quad \wedge tr \upharpoonright \alpha Q \in F_{TT}(Q) \wedge \forall (a, t_a) \in tr \upharpoonright \alpha P \cap \alpha Q \\
&\quad \wedge t_a \in I_{a,P} \cap I_{a,Q} \neq \emptyset\} \\
F_{TT}(P ||| Q) &= \{\langle\rangle\} \cup \{tr \in M_{TT} \mid tr \upharpoonright \alpha P \in F_{TT}(P) && (3.4.9)\\
&\quad \wedge tr \upharpoonright \alpha Q \in F_{TT}(Q)\}
\end{aligned}
$$

1. Cuando el proceso $t_0.\star \rightarrow P$ ofrece el evento de instanciación $\star$, se pueden dar dos situaciones diferentes:

   a) no se consume nunca el evento $\star$, eso significa que el entorno no llega nunca a ofrecerlo; en ese caso la ejecución del proceso es una traza vacía.

b) el entorno ofrece el evento$\star$ y el proceso lo consume en el tiempo t_0, en este caso, la ejecución del proceso es la concatenación del evento temporizado $(\star, t_0)$ con las trazas temporizadas de P.

2. Las trazas de la ejecución del proceso $e \bowtie v \rightarrow P$ sigue la misma lógica de las trazas del proceso $t_0.\star \rightarrow P$.

   Por ejemplo, las posibles trazas temporizadas del proceso $1.\star \rightarrow a \bowtie t \rightarrow P$ son: $\langle\rangle$, $\langle(\star, 1)\rangle$, $\langle(\star, 1)^\frown(a, t \geq 1)\rangle$, (tal y cómo se ve en la figura 3.4).

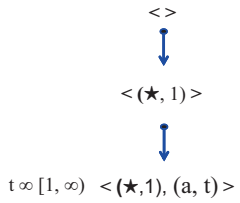


Figura 3.4: Traza 1

3. Si el evento e no se ofrece nunca por el entorno en el intervalo $I(T, t)$, entonces el proceso $I(T, t).e \rightarrow P$ no puede consumir ningún evento, por lo cual su ejecución va a ser equivalente a una traza vacía. Sin embargo, si el entorno ofrece el evento dentro de su intervalo de habilitación, el proceso consume el evento y sigue ejecutando el proceso P. Por ejemplo, las posibles trazas temporizadas del proceso $0.\star \rightarrow I(5, 0).e \rightarrow STOP$, son: $\langle\rangle$, $\langle(\star, 0)\rangle$, $\langle(\star, 0)^\frown(e, 1 \leq t \leq 6)\rangle$, (tal y cómo se ve en la figura 3.5).

   se comporta de la siguiente manera:

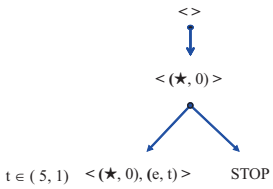


Figura 3.5: Traza 2

4. El proceso $I(T, t) \rightarrow P$ retrasa la activación del proceso P hasta que se vence el intervalo de tiempo $I(T, t)$. Si el evento de la terminación

exitosa ($\surd$) no se ofrece nunca por el entorno, el evento P no llegaría nunca a ejecutarse: en caso contrario, el proceso consume el evento $\surd$ y pasa a ejecutar el proceso P.

5. El proceso $P \ \square \ Q$ se comporta como P o como Q dependiendo del primer evento visible que se ofrece (dentro de su intervalo de habilitación) por el entorno, entonces, se ejecuta el proceso P o el proceso Q.

6. El proceso $P \ \sqcap \ Q$ se comporta como P o Q dependiendo de la elección interna del proceso, entonces su ejecución es simplemente las observaciones que acompañan el proceso P o el proceso Q.

7. La ejecución del proceso $P; Q$ son las observaciones que acompañan la ejecución del proceso P, seguida con las observaciones de Q

8. El proceso $P \underset{B}{\|} Q$, la ejecución proyectada en el conjunto de eventos αP y la ejecución proyectada en el conjunto de eventos αQ son resultado de la ejecución de P y Q respectivamente. Los procesos P y Q deben sincronizarse en los eventos del conjunto B. Un evento $a \in B$ observable en la ejecución de $P \underset{B}{\|} Q$ es el resultado de que los dos procesos P y Q aceptan consumir el evento, además de que la intersección del intervalo de activación del evento a en P ($I_{a,P}$) y el intervalo de activación del mismo en $Q(I_{a,Q})$ es diferente del conjunto vacío, y el evento se consume por el proceso $P \underset{B}{\|} Q$ en un tiempo $t_a \in I_{a,P} \cap I_{a,Q}$.

9. El proceso $P|||Q$, la ejecución proyectada en el conjunto de eventos αP y la ejecución proyectada en el conjunto de eventos $(\alpha Q)^{\surd}$ son resultado de la ejecución de P y Q, respectivamente.

3.4.3. Rechazos temporizados

Los sistemas de tiempo real son sistemas reactivos, para ello, es importante saber cuándo un proceso interacciona con su entorno, y cuándo no es posible esta interacción. En los sistemas deterministas, está información se puede obtener de la traza temporizada de un proceso,sin embargo en los sistemas no deterministas, esta información no es suficiente. Por ejemplo, La trazas

temporizadas de los dos procesos

$$a \rightarrow STOP \quad y \quad STOP \sqcap a \rightarrow STOP$$

son idénticas, pero el primer proceso debe responder siempre en un entorno donde se ofrece el evento a, mientras que el segundo puede negar a responder.

De ello, se debe registrar, la información de *los rechazos temporizados* también. Los rechazos temporizados contienen los eventos temporizados que el proceso rechaza durante una ejecución, junto con los periodos donde se rechazan estos eventos.

Definimos formalmente el conjunto TRF de la información rechazada como sigue:

$$TRF = \{(\aleph | \exists (t_i, \tau_i) \in \Re^+ \times \Re^+; t_i \leq \tau_i; A_i \in \Sigma;\ \aleph = \bigcup_{1..n} ([t_i, \tau_i) \times A_i))\}$$

Definimos algunas operaciones en estos objetos. Sea $\aleph \in TRF$ y $t_1, t_2 \in \Re^+, A \subseteq \Sigma$, y $a \in \Sigma$
$\emptyset \in \aleph$
$\aleph \uparrow [t_1, t_2) = \aleph \cap [t_1, t_2) \times \Sigma$
$\aleph \restriction_i t = \aleph \restriction [0, t)$
$\aleph \restriction A = \aleph \cap [o, \infty) \times A$
$\varphi(\aleph) = \{a | \aleph \restriction a \neq \emptyset\}$

$begin(\aleph) = inf(\{t | \exists a, (t, a) \in \aleph\} \cup \infty)$
$end(\aleph) = sup(\{t | \exists a, (t, a) \in \aleph\} \cup 0)$
$\aleph + t$, incrementa con t todos los tiempos asociados a los eventos en $\aleph$

3.4.4. Modelos de fallos temporizados

En las aplicaciones, donde solamente las propiedades de *seguridad* y *vivacidad* se deben especificar, el modelo de *fallos temporizados* es adecuado.

En este modelo el proceso está asociado con el conjunto de trazas temporizadas y el conjunto de rechazos temporizados. Formalmente, los fallos temporizados es el conjunto TF, tal que

$$TF = \{(s, \aleph) | s \in M_{TT} \quad \wedge \quad \aleph \in TRF\}$$

Los fallos temporizados $(s, \aleph)$ se interpretan como una observación del proceso, donde los eventos que el proceso realiza están registrados en s, mientras que los eventos rechazados junto con los intervalos donde han sido rechazados, se registran en $\aleph$. Por ejemplo, el proceso P que realiza el evento, a y b en un tiempo 1 y 2, respectivamente, y rechaza el evento b hasta el tiempo 2, incluye el par $(((a,1),(b,2)),[0,2) \times b)$ en su definición semántica. Para ello, dos proceso se consideran diferentes, si existe diferencia en sus trazas temporizadas o, diferencia en sus rechazos temporizados. Llamamos el par de trazas temporizadas y rechazos temporizados, el *comportamiento* del proceso.
Definimos algunas operaciones en este objeto:

Sea $s \in M_{TT}$ y $\aleph \in TRF$, $t, t' \in \Re^+$, $A \subseteq \Sigma$ y $a \in \Sigma$

$$(s, \aleph) \restriction t = (s \restriction t, \aleph \restriction t)$$

$$(s, \aleph) \restriction_i t = (s \restriction_i t, \aleph \restriction_i t)$$

$$(s, \aleph) \restriction a = (s \restriction a, \aleph \restriction a)$$

$$begin((s, \aleph)) = min\{begin(s), begin(\aleph)\}$$

$$end((s, \aleph)) = max\{(begin(s), begin(\aleph)\}$$

$(s_1, \aleph_1)$ **in** $(s_2, \aleph_2) \Leftrightarrow \exists s_3 \in ttr$ *tal que* $s_1 = s_2 \frown s_3$ *y* $\aleph_2 \subseteq \aleph_1 \restriction begin(s_2)$
$(s_1, \aleph_1)$ **notin** $(s_2, \aleph_2) \Leftrightarrow \nexists s_3 \in ttr$ *tal que* $s_1 = s_2 \frown s_3$ *y* $\aleph_2 \subseteq \aleph_1 \restriction begin(s_2)$

El modelo de las fallas temporizadas M_{FT}, es el espacio de todos los subconjuntos $P \subseteq TF$. Los fallos temporizados de un proceso representan sus posibles aspectos observables. Definimos la función semántica $F_{FT} : CSP + T \longmapsto M_{FT}$, que mapea cada proceso a su posible conjunto de fallos.

$F = F_{FT}(P)$ satisface las siguientes propiedades:

1. (P1): $(\langle\rangle, \emptyset) \in F$
2. (P2): $(s, \aleph \in F \wedge (s_1, n_1)$ **in** $(s, \aleph)) \Rightarrow (s_1, \aleph_1) \in F$
3. (P3): $\forall t \geq 0, \exists n \in N; (s, \aleph) \in F \ \wedge \ end(s) \leq t) \Rightarrow; \#s \leq n$
4. (P4): $(s, \aleph) \in F \wedge \forall t \geq 0, (a, t) \notin \aleph \Rightarrow (s \restriction_i t \frown (a, t), \aleph \restriction_i t) \in F$

- La propiedad (P1) significa que el proceso puede no aceptar ni rechazar ningún evento.
- La propiedad (P2) indica que cualquier comportamiento observable, se puede reemplazar con una especificación más abstracta y con menos información.
- (P3) indica que un proceso no puede realizar infinitamente muchos eventos en un tiempo finito.
- (P4) implica que si un evento a no se rechaza por un proceso en un tiempo t, se puede haber realizado en este tiempo.

$$
\begin{aligned}
F_{FT}(STOP) &= \{(\langle\rangle\,,\aleph)\,|\,\aleph \in TRF\} && (3.4.10)\\
F_{FT}(SKIP) &= \{(\langle\rangle\,,\aleph)\,|\,\surd \notin \varphi(\aleph)\} && (3.4.11)\\
&\quad \cup \\
&\quad \{(\langle(\surd,t)\rangle\,,\aleph)\,|\,t \geq 0 \wedge \surd \notin \varphi(\aleph \restriction_i t)\}\\
F_{FT}(t_0.\star \rightarrow P) &= \{(\langle\rangle\,,\aleph)\,|\,\star \notin \varphi(\aleph)\} && (3.4.12)\\
&\quad \cup \\
&\quad \{((\star,t_0)^\frown s,\,\aleph)\,|\,s \in F_{TT}(P)\\
&\quad \wedge\, 0 \leq t_0 \leq begin(s) \wedge \star \notin \aleph \uparrow 0,\,t_0)\}\\
F_{FT}(I(T,\,t).e \bowtie v \rightarrow P) &= \{(\langle\rangle\,,\aleph)|\ \ e \notin \varphi(\aleph \uparrow (T,\,t))\} && (3.4.13)\\
&\quad \cup \\
&\quad \{((e,\,v)^\frown s\ \aleph)\,|\,s \in F_{TT(P)}\\
&\quad \wedge\, t \leq v < t+T \wedge begin(s) \geq v\\
&\quad \wedge\, e \notin \aleph \uparrow [t,\,v]\}\\
F_{FT}(I(T,t) \rightarrow P) &= (\langle\rangle\,,\aleph) && (3.4.14)\\
&\quad \cup \\
&\quad \{((\surd \frown s) + (t+T),\,\aleph + (t+T))\\
&\quad |\,(s,\,\aleph) \in F_{FT}(P)\}
\end{aligned}
$$

$$
\begin{aligned}
F_{FT}(P \underset{B}{\|} Q) \;=\; & \{(s, \aleph) \mid \exists\, \aleph_1, \aleph_2; \\
& \aleph \restriction (\Sigma - B) \subseteq (\aleph_1 \restriction (\Sigma - B) \\
& \cap\, \aleph_2 \restriction (\Sigma - B)) \\
& \wedge\, \aleph \restriction (B) \subseteq (\aleph_1 \restriction (B) \cap \aleph_2 \restriction (B) \\
& \wedge\, s = s \restriction (\alpha(A) \cup \alpha(B))) \\
& \wedge\, (s \restriction \alpha(A),\, \aleph_1) \in F_{FT}(P) \\
& \wedge\, (s \restriction \alpha(B),\, \aleph_2) \in F_{FT}(Q))\}
\end{aligned}
\tag{3.4.15}
$$

$$
\begin{aligned}
F_{FT}(P|||Q) \;=\; & \{(s, \aleph) \mid \exists\, \aleph_1, \aleph_2; \\
& \aleph \restriction (\Sigma - B) \subseteq (\aleph_1 \restriction (\Sigma - B) \\
& \cap\, \aleph_2 \restriction (\Sigma - B)) \\
& \wedge\, s = s \restriction (\alpha(A) \cup \alpha(B))) \\
& \wedge\, (s \restriction \alpha(A),\, \aleph_1) \in F_{FT}(P) \\
& \wedge\, (s \restriction \alpha(B),\, \aleph_2) \in F_{FT}(Q))\}
\end{aligned}
\tag{3.4.16}
$$

1. El proceso $STOP$ no puede involucrarse en ningún evento; asimismo, no puede aceptar ni rechazar ningún evento.

2. El proceso $SKIP$ se puede entender como el proceso $\surd \rightarrow STOP$.

   Si el evento $(\surd)$ se ofrece por el entorno el proceso debe aceptarlo, es decir, si $F_TT(SKIP) = \langle\rangle$, entonces $\surd$ no se ha ofrecido por el entorno, y obviamente, no debe pertenecer a los elementos del conjunto de rechazos del proceso $\aleph$.

   En caso contrario, el proceso acepta el evento $(\surd)$ y continua comportándose como el proceso $STOP$. Si el proceso termina en un tiempo t, significa que el entorno no podía ofrecer el evento de terminación antes, por lo cual $\surd$ no debe aparecer en el conjunto de rechazos antes del tiempo t $(\surd \notin \varphi(\aleph \restriction_i t))$

3. El proceso $t_0.\star \rightarrow P$ se puede comportar de dos maneras diferentes:

   (1) El entorno no llega a ofrecer el evento $\star$, entonces el proceso se comporta como una traza vacía rechazando todos los eventos que le

podía ofrecer el entorno. Obviamente, el evento $\star$ no puede aparecer en el conjunto de rechazos del proceso.

(2) Si el proceso se instancia en un tiempo t_0, significa que el entorno no podía ofrecer el evento de instanciación en un tiempo anterior a t_0, por lo cual el evento $\star$ no puede pertenecer al conjunto de rechazos proyectado en el intervalo de tiempo $[0,\, t_0]$ ($\star \notin \varphi(\aleph \restriction_i t_0)$). Además, al instanciarse, el proceso pasa a comportarse como P. El primer evento en la traza temporizada $s \in F_T(P)$ debe ocurrir en un tiempo superior o igual a 0, o sea, $\forall s \in F_{TT}(P), begin(s)) \geq 0$.

4. El proceso $I(T,\, t).e \bowtie v \rightarrow P$ sólo puede aceptar involucrarse en el evento e, si el entorno se lo ofrece dentro del intervalo de tiempo $I(T,\, t)$[11]. Este proceso puede comportarse de dos maneras diferente:

   (1) El entorno nunca le ofrece el evento e o se lo ofrece fuera del intervalo de tiempo $I(T,\, t)$, entonces el proceso no realiza ningún evento y rechaza todos los eventos que le ofrece el entorno. En este caso, el evento e no puede pertenecer al conjunto de eventos proyectado al intervalo $I(T,\, t)$, dado que el evento no se ofrece en este intervalo y, de lo contrario, el proceso estaría obligado a aceptarlo.

   (2) El entorno ofrece el evento e en un tiempo v, tal que $t \leq v < t + T$, y continua comportándose como P. El tiempo de ocurrencia del primer evento en la traza temporizada de P debe ser superior a v ($begin(s) \geq v$). Es importante mencionar que el evento e no puede pertenecer al conjunto de rechazos del procesos proyectado al intervalo de tiempo $[t,\, v]$, ya que si se ofrece en dicho intervalo, el proceso lo hubiera aceptado.

5. El proceso $I(T,\, t) \rightarrow P$ tiene el mismo comportamiento de P, retrasado con un tiempo $t + T$.

6. La elección $P \,\Box\, Q$ se resuelve en favor de uno de los dos procesos P o Q.

   (1) Antes de la resolución de la elección [12], todos los eventos rechazados por el proceso, $P \,\Box\, Q$ son eventos rechazados por los dos procesos P y Q. Es decir, el comportamiento del proceso antes de la resolución de la

[11] El intervalo $I(T,\, t)$ es equivalente al intervalo $I[t,\, t + T]$

[12] Es decir, antes de que el proceso $P \,\Box\, Q$ no se involucre en ningúno de los eventos $head(s)$ o $head(\acute{s})$ tal que, $s \in F_{TT}(Q)$ y $\acute{s} \in F_{TT(P}$

elección es equivalente al comportamiento de los dos procesos, P y Q. Observamos que:

$$F_{FT}(P \,\square\, Q) \restriction_i t_0 = F_{FT}(P) \restriction_i t_0 = F_{FT}(Q) \restriction_i t_0,$$

con $t_0 = begin(F_{TT}(P \,\square\, Q))$

(2)Después de la resolución de la elección, el proceso $P \,\square\, Q$ se comporta como P si el primer evento ocurrido es $head(F_{TT}(P))$ o como Q si el primer evento ocurrido es $head(F_{TT}(Q))$, es decir, el proceso $P \,\square\, Q$ se comporta como $F_{FT}(P)$ o como $F_{FT}(Q)$.

3.4.5. Especificación de propiedades con el modelo de trazas temporizadas

Las propiedades que debe cumplir un proceso $CSP+T$, se define en términos de trazas temporizadas. Esta definición, caracteriza las trazas que son aceptable y las trazas que no son aceptables. Un proceso cumple con su especificación si toda ejecución del mismo es aceptable, es decir, ninguna ejecución del sistema viola la especificación. Asimismo, si $S(tr)$ es un predicado en la traza temporizada tr, decimos que P *satisface o cumple* con $S(tr)$, si $S(tr)$ se mantiene por cualquier traza temporizada $tr \in F_{TT}(P)$.

$$P \mathbf{sat} S(tr) \Leftrightarrow \forall tr \in F_{TT}(P); S(tr)$$

Por ejemplo, en una máquina de café automática, no se puede parar la máquina hasta que transcurra 5 unidades de tiempo desde el instante de su puesta en marcha.

$$S(tr) = \forall v \in R^+, (on, v) \textbf{ in } s \Rightarrow off \notin (\varphi(s \uparrow [v, v+5]))$$

El proceso:

$$P = on \bowtie v \rightarrow (5, v) \rightarrow off \rightarrow STOP$$

tiene el conjunto de trazas temporizadas:

$$F_{TT}(P) = \{\langle\rangle \ \langle (on, v) \rangle, \langle (on, v), (off, v \leq t \leq v+5) \rangle\}$$

y cualquier traza tr en $F_{TT}(P)$ satisface $S(tr)$, es decir,

$$P \ \ \textbf{Sat} \ \ \forall v \in R^+, (on, v) \ \textbf{in} \ s \Rightarrow off \notin (\varphi(s \uparrow [v, v+5])$$

La especificación de un sistema en el modelo de trazas temporizadas o en el modelo de trazas en general, permite la expresión de las propiedades o condiciones **de seguridad**, estas propiedades requieren que "*nada malo pase*". En nuestro contexto *algo malo* es una traza temporizada que no satisface $S(tr)$. De ello, la especificación de una propiedad de seguridad se puede traducir a una prohibición de la ocurrencia de una traza temporizada s en la ejecución de un proceso P, o también, la prohibición de la ocurrencia de un evento en un tiempo determinado $\forall tr \in F_{TT}(P)$.

$$S(tr) = s \ \textbf{notin} \ tr$$

Por ejemplo, en un sistema robotico, si la planta de la prensa se mueve ($move_{up}$) antes de que el brazo del robot que estaba depositando una pieza en ella, se haya girado ($turn$), puede ocurrir una colisión entre estos dos componentes. Una propiedad que debe tener el sistema para descartar está colisión se expresa de la siguiente manera:

$$\mathbf{S(tr)} = \forall t_1, t_2 \in R^+, \langle (move_{up}, t1)^\frown (turn, t2) \rangle \ \textbf{notin} \ tr$$

El proceso

$$\mathbf{Sys} = deposit \rightarrow turn \bowtie t_c \rightarrow STOP || I(5, t_c).move_{up} \rightarrow STOP$$

La traza temporizada del proceso Sys es:

$$\mathbf{F_{TT}(Sys)} = \{\langle\rangle, \langle (deposit, t_d) \rangle, \langle (deposit, t_d)^\frown (turn, t_c) \rangle,$$
$$\langle (deposit, t_d)^\frown (turn, t_c)^\frown (move_{up} \leq t \leq t+5) \rangle\}$$

De ello, $\forall tr \in F_{TT}(Sys)$ y $\forall t_1, t_2 \in R^+$,

$$\langle (move_{up}, t1)^\frown (turn, t2) \rangle \ \textbf{notin} \ tr$$

3.4.6. Especificación de propiedades con el modelo de fallos temporizados

Una propiedad en el modelo de fallos se describe en termino de restricciones en las trazas temporizadas y restricciones en los rechazos temporizados. Se dice que un proceso P cumple con la especificación $S(s, \aleph)$, si S se mantiene por cualquier ejecución $(s, \aleph) \in F_F T(P)$. Ésto se escribe, P **sat** $S(s, \aleph)$.

$$P \textbf{ sat } S(s, \aleph) \Leftrightarrow \forall (s, \aleph); S(s, \aleph)$$

La especificación $S(s, \aleph)$ se suele expresar en términos de proyección de la traza y rechazos sobre un conjunto de eventos o intervalos de tiempo.

La especificación de un sistema en el modelo de fallos temporizados permite además de la descripción de propiedades de **seguridad**, la descripción de las propiedades de **vivacidad**.
Las propiedades de vivacidad especifican que "*algo bueno sucederá*", eventualmente, durante la ejecución del sistema. En nuestro contexto, *algo bueno* es la involucración exitosa en todos los eventos de una traza temporizada tr. Éste se puede entender como la inhabilidad de rechazar eventos en situaciones particulares, o sea, exigir al proceso involucrarse en estos eventos.

Por ejemplo, el sistema robótico, debe estar preparado para ponerse en marcha. Este requerimiento se puede expresar como una propiedad de vivacidad asociada a la traza vacía.

$$S(s, \aleph) = (\ \ s = \langle\rangle \Rightarrow on \notin \varphi(\aleph))$$

Esta condición indica si ningún evento ha sucedido todavía, el evento on no se puede rechazar si se ofrece por el entorno.

por ejemplo, el proceso $P = on \rightarrow P'$, teniendo como traza temporizada la traza vacía, no puede rechazar el evento on, si se ofrece por el entorno. De ello, $P = on \rightarrow P'$ **sat** $s = \langle\rangle \Rightarrow on \notin \varphi(\aleph)$.

3.4.7. Refinamiento de términos de proceso CSP+T

La relación de refinamiento se puede describir en CSP+T en diferentes maneras, depende del modelo semántico que se está usando. Dos formas relevantes

de refinamiento, corresponden a los dos modelos presentados anteriormente: Modelo de trazas temporizadas y modelos de fallos temporizados.

En semántica de trazas temporizada: se dice que una especificación P satisface o refina a un especificación P_0 ($P \models_T P_0$), si toda traza de P es permitida por P_0, es decir, toda posible secuencia de comunicaciones que puede realizar P, es también posible por P_0.

$$P \models_T P_0 \Leftrightarrow F_{TT}(P_0) \subset F_{TT}(P)$$

En semántica de fallos temporizados: si P es una implementación de un sistema que cumple con su especificación P_0. Si cada posible observación realizada sobre una ejecución de P_0 puede también aparecer en la la observación de P, es decir,

$$P \models_F P_0 \Leftrightarrow F_{FT}(P_0) \subseteq F_{FT}(P)$$

Capítulo 4

Proceso de Transformación de UML–RT a CSP

4.1. Introducción

En este capítulo se va a explicar el proceso de transformación que se ha seguido para integrar el lenguaje formal CSP+T al modelado de sistemas de tiempo real utilizando UML–RT desde el marco metodológico MEDISTAM–RT. Se exponen los metamodelos de cada uno de los diagramas de representación admitidos dentro de MEDISTAM-RT en el plano semiformal, y luego se detalla el proceso de transformación seguido.

4.2. Integración de lenguajes formal y semiformal en MEDISTAM–RT

MEDISTAM–RT [BCH06, BCHM07] proporciona un marco metodológico para la correcta especificación de sistemas de tiempo real combinando lenguajes

semiformales basados en UML con el lenguaje formal CSP+T. Esta combinación de lenguajes se realiza con la intención de aprovechar la potencia expresiva de UML, así como de los perfiles definidos para este lenguaje, en procesos de Desarrollo de Software Dirigido por Modelos (MDSD, del inglés *Model-driven Software Development*). Para ello, la metodología proporciona una serie de metamodelos que definen los conceptos de cada diagrama en MEDISTAM-RT, y las relaciones entre ellos (p.e., eventos temporizados, intervalos de habilitación, cápsulas, protolos, etc.). Los modelos en UML que instancian los metamodelos MEDISTAM-RT, pueden ser transformados en especificaciones formales de procesos en CSP+T aplicando un conjunto de reglas de transformación completo que propone la metodología.

Esta correspondencia entre especificaciones en ambos lenguajes, UML y CSP+T, permite disponer de un modo indirecto de modelos que describen distintos aspectos del sistema en un modelo matemático, y de forma global de una especificación completa del sistema. La especificación puede ser verificada utilizando técnicas de verificación en distintos niveles de abstracción. Y cómo se verá en el siguiente capítulo, las reglas de transformación posibilitan tener una doble visión de la especificación del sistema desde el enfoque formal como del enfoque semiformal, por lo que el proceso de refinamiento de algún modelo en el lado semiformal conlleva un refinamiento en la especificación del sistema en el lado formal.

4.2.1. Aproximaciones para la integración de un lenguaje formal

Existen dos posibles estrategias de integración de los métodos formales en el proceso del desarrollo de los sistemas que utilizan las representaciones semiformales como punto de partida para obtener especificaciones formales [FKV94]:

- **Aproximación paralela:** en esta aproximación las dos representaciones, semiformal y formal, se elaboran de manera simultánea con refinamientos sucesivos. Esta aproximación permite pasar, en cualquier momento durante su especificación, de una representación semiformal a su correspondiente representación formal, y viceversa. De esta manera, cada formalismo podría enriquecer los aspectos del otro.

- **Aproximación secuencial:** empieza con una descripción completa del sistema en un lenguaje semiformal y luego se realiza la transformación a una especificación formal. En esta aproximación no se permiten vueltas hacia la representación semiformal.

Para poder llevar a cabo la integración distinguimos dos posibles caminos:

- **Integración por adjunción:** esta aproximación consiste en proveer semántica formal a métodos semi-formales, enriqueciendo su sintaxis con nuevos conceptos que los permita ser usados de una manera formal.
- **Integración por derivación:** consiste en construir primero los modelos gráficos y luego obtener su equivalente en un lenguaje formal aplicando una serie de reglas de transformación que deben de ser establecidas previamente para este propósito.

En MEDISTAM–RT hemos combinado las dos aproximaciones anteriores de tal modo que se han añadido conceptos inspirados en el método formal CSP+T a los diagramas de UML para ayudar al desarrollador a modelar de forma intuitiva modelos más formales, sacrificando en parte la flexibilidad que suelen permitir las representaciones semiformales. Por otra parte, se ha de obtener una especificación formal del sistema que pueda ser usada para la validación y verificación automática del sistema, lo que requiere llevar a cabo un proceso de transformación. Por tanto, también se ha seguido un proceso de integración por derivación para obtener la especificación en CSP+T a partir de los modelos UML construidos por adjunción de conceptos CSP+T. Como el lenguaje UML está diseñado con una arquitectura de capas, la técnica de la derivación por integración se puede definir a diferentes niveles de la arquitectura de UML:

1. *Derivación a partir de modelos:* En este caso, se tiene que establecer una correspondencia entre cada elemento del modelo semiformal a construcciones formales [FA00].
2. *Derivación a partir de Metamodelo:* consiste en establecer una correspondencia entre los elementos del metamodelo a construcciones formales, es decir, formalizar el metamodelo de UML.

3. *Derivación a partir de meta-meta-modelo:* En este caso, se formaliza el metametamodelo y también las reglas de instanciación del metamodelo UML [FA01].

Estas tres técnicas son diferentes y no tienen el mismo objetivo.

4.2.1.1. Transformación de modelos UML–RT a procesos CSP+T

La especificación rigurosa y el razonamiento sobre las propiedades de un sistema, requiere la utilización de los métodos formales desde las primeras etapas del desarrollo del sistema. En nuestra metodología combinamos los modelos del UML–RT y CSP+T. La idea de acoplar los modelos formales y semi-formales no es reciente y fué fuertemente impulsada durante los años 90 y es conocida como aproximación mixta. En esta tesis presentamos un método que consiste en la generación de modelos formales CSP+T a partir de representaciones semi-formales UML–RT.

Obviamente, la forma de pasar de un lenguaje semi-formal hacia otro formal debe ser específica, consistiendo en unas reglas de transformación [1]. Esto implica, al menos, la definición de un conjunto de reglas que expliquen cómo los elementos de modelado semi-formales son trasladados sobre el modelo matemático formal.

Dentro del marco de la metodología MEDISTAM–RT se utilizan varios tipos de modelos utilizando una descripción gráfica semiformal que se representan en los siguientes tipos de diagramas:

- Diagramas de estado temporizados
- Diagramas de Secuencia Temporizados
- Diagramas de Clase
- Diagrama de Estructura compuesta

Nuestro objetivo se centra en darles una semántica formal a estos modelos que los haga adecuados a la especificación de los STR, y al mismo tiempo

[1] Son reglas de transformación y no como podría interpretarse en un proceso de traducción, ya que las representaciones formales y semi-formales, por definición, no son matemáticamente equivalentes.

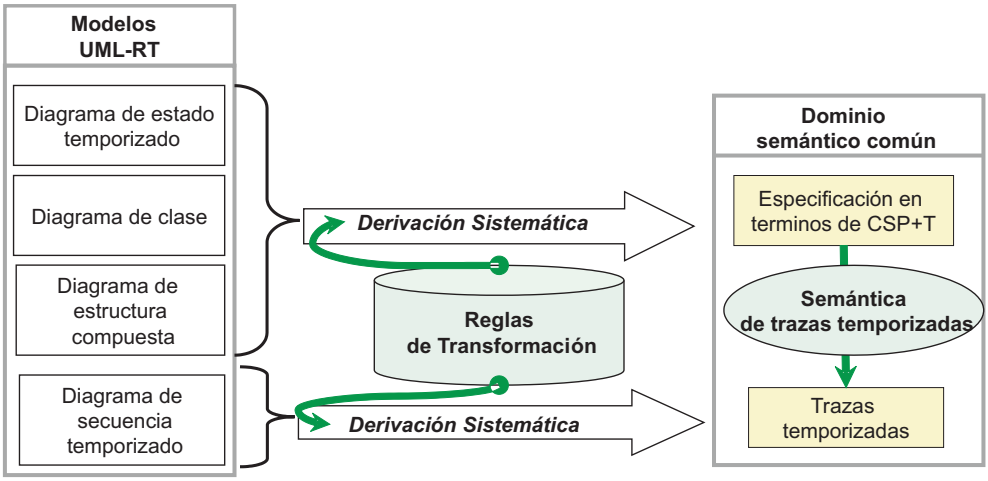


Figura 4.1: Transformación de modelos de UML–RT

llevarlos a un dominio semántico común, para poder razonar sobre la consistencia entre las diferentes vistas que representan. Asimismo, como se muestra en la figura 4.1, se define un conjunto de reglas de transformación para cada uno de los diagramas de estados temporizados, de clase y de estructura compuesta de tal forma que su aplicación permite derivar sistemáticamente una especificación formal en términos de CSP + T, y también se define un conjunto de reglas que transforma diagramas de secuencia a modelos de trazas temporizadas.

En las siguientes secciones vamos en primer lugar a describir de manera informal todos los conceptos involucrados en el marco metodológico de MEDISTAM–RT utilizando lenguajes semiformales, para posteriormente dotarles de una semántica formal.

4.3. Modelado en el plano semiformal

MEDISTAM–RT define un método para el diseño correcto de sistemas de tiempo real que se fundamenta en el desarrollo de modelos desde un plano semiformal siguiendo un enfoque metodológico que se verá con más detalles

en el siguiente capítulo, cubriendo los aspectos estructurales y de comportamiento del sistema, así como los aspectos temporales del mismo. Los modelos desarrollados utilizando la descripción semiformal de UML y UML–RT no es suficiente para luego poder trasladarlo al ambiente formal de CSP+T. Por tanto, necesitamos modificar la semántica de los metamodelos de UML para incluir los conceptos y semántica adicionales necesarios utilizando un enfoque de integración por adjunción que, posteriormente, nos permita derivar a partir de los modelos la especificación del sistema en el plano formal.

En esta sección analizaremos los metamodelos específicos de UML dentro del contexto de MEDISTAM–RT, examinando los conceptos y la semántica asociada en cada uno de los diagramas de representación definidos.

La estructuración de los modelos de MEDISTAM–RT se engloban desde dos vistas:
Vista estática: Permite describir la estructura y propiedades interna del sistema bajo dos perspectivas:

- *Estructural:* describe el contenido del componente, cúales son sus partes y cómo están relacionadas.
- *De datos:* describe los datos manipulados en el componente.

Esta vista se modela con los dos diagramas complementarios, el diagrama de clase y el diagrama de estructura compuesta que se describen en detalle en la sección 4.3.1.
Vista dinámica y temporal: permite visualizar el sistema como caja negra, o también como caja blanca. Estas diferentes perspectivas permiten distinguir el comportamiento externo del comportamiento interno de una cápsula o sistema.

- *Comportamiento interno temporizado:* describe como se comporta un componente para proporcionar servicios o realizar tareas.
- *Comportamiento externo temporizado:* este aspecto concierne las interacciones del componente con su entorno, asimismo provee una vista caja negra que solo revela una abstracción de los servicios proporcionados o requeridos por el componente vía sus controladores en el transcurso de tiempo. Por consecuente, esta vista representa una abstracción del comportamiento externo temporizado donde se ocultan los eventos internos del componente.

Esta vista se modela con los dos diagramas: diagrama de estado temporizado y diagrama de secuencia temporizado que se presentan a continuación en la sección 4.3.2.

4.3.1. Vista estructural

4.3.1.1. Diagrama de clases en MEDISTAM–RT

El diagrama de clase describe el aspecto estructural de un sistema. Para cubrir el aspecto estructural del mismo, define los elementos de un sistema y la relaciones entre ellos. Adicionalmente, visualiza los eventos que se intercambian las cápsulas del sistema para comunicarse y también, atributos y operaciones internas de cada elemento, para cubrir el aspecto de datos. Este diagrama no muestra la información temporal. Los diagramas de clases facilitan el diseño jerárquico en MEDISTAM-RT. En la figura 4.2, se muestra el metamodelo del diagrama de clase en MEDISTAM-RT.

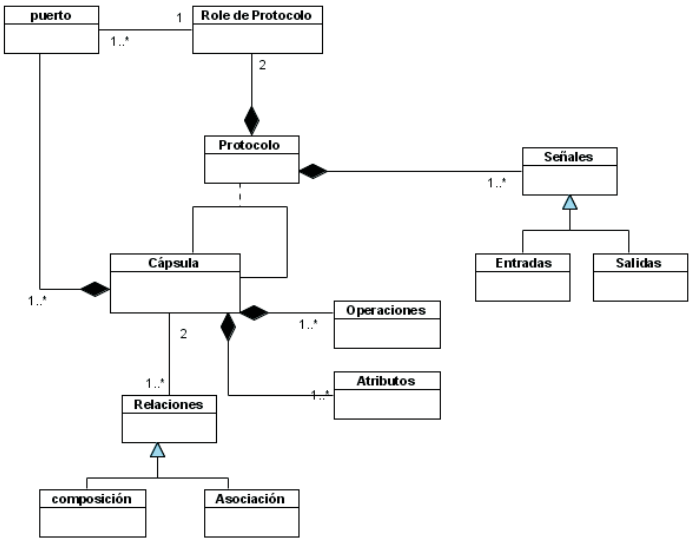


Figura 4.2: Metamodelo del diagrama de clase de MEDISTAM-RT

A continuación se describen los elementos que componen este metamodelo.

Conceptos	**Definiciones**
Cápsula	Es una parte modular del sistema, que puede ser considerada como una unidad que funciona de manera autónoma dentro del sistema.
Señales	Son estímulos o eventos que representan un medio de comunicación entre dos cápsulas.
Puerto	Son elementos a través de los que una cápsula recibe o envía una señal desde o al exterior.
Protocolo	Encapsula las comunicaciones potenciales entre componentes.
Rol de protocolo	Es una combinación de la interfaz requerida y la interfaz proporcionada.
Eelaciones	Son relaciones que se pueden establecer entre las cápsulas.
Operaciones	Son métodos internos de una cápsula

Cápsula: una cápsula en MEDISTAM–RT se define como una parte modular del sistema que funciona de manera autónoma dentro del sistema, de forma equivalente a la definición de un componente en UML 2.0. Una cápsula en MEDISTAM–RT está compuesta de dos partes: interfaces y contenidos.

Señales: una señal es un estímulo o una comunicación entre dos cápsulas que transmiten información con la esperanza de que una acción tenga lugar.

Operaciones: son métodos que lleva a cabo un cápsula de manera interna sin interaccionar con el exterior.

Puertos: son elementos a través de los que una cápsula envía o recibe señales. El conjunto de señales que se pueden pasar por un puerto se encapsulan en los protocolos.

Protocolo y Rol de protocolo: un protocolo encapsula el conjunto de señales o comunicaciones intercambiadas entre dos o más componentes. Un protocolo comprende un conjunto de puertos, donde cada uno de ellos juega un rol especifíco en el protocolo. Un caso particular es *el protocolo binario* que involucra solamente dos puertos, por la que su especificación es más sencilla.

La ventaja del ***protocolo binario*** frente a otros es que sólo tiene dos roles; uno básico y otro conjugado, de tal manera que solo se necesita especificar el rol básico; el rol conjugado puede derivarse del básico con simplemente invertir el conjuntos de las señales de entrada por las de salida y vice versa.

Por ejemplo, en la figura 4.3, el emisor y el receptor son dos roles del mismo protocolo. Los mensajes de entrada del emisor representan las salidas del receptor, y los mensajes de salida del emisor corresponden a las de entrada del receptor.

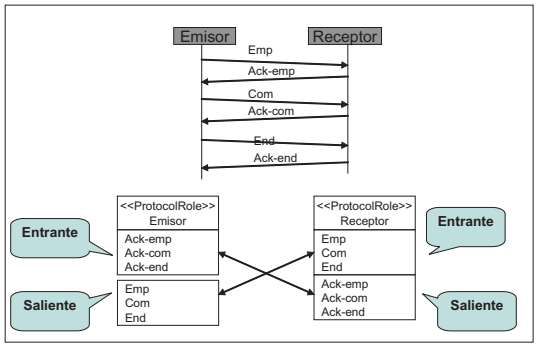


Figura 4.3: Protocolo

Relaciones: dos tipos de relaciones se pueden establecer entre los cápsulas, estos son: las asociaciones y las agregaciones. La asociación entre dos cápsulas se modela con un protocolo común agregado a estos dos cápsulas. Las agregaciones muestran una relación de pertenencia fuerte entre una cápsula y su parte.

4.3.1.2. Diagrama de estructura compuesta

El diagrama de estructura compuesta se representa, desde el punto de vista estructural la organización global del sistema, como una composición de componentes que se conectan entre sí. El diagrama de estructura compuesta facilita el diseño arquitectónico del sistema dentro de MEDISTAM–RT. En la figura 4.7 se muestra el metamodelo que visualiza los elementos de modelado que incluye el diagrama de estructura compuesta. A continuación se definen cada

uno de los elementos de modelado y su semántica, y por otro lado se puede ver la relación que hay entre ellos.

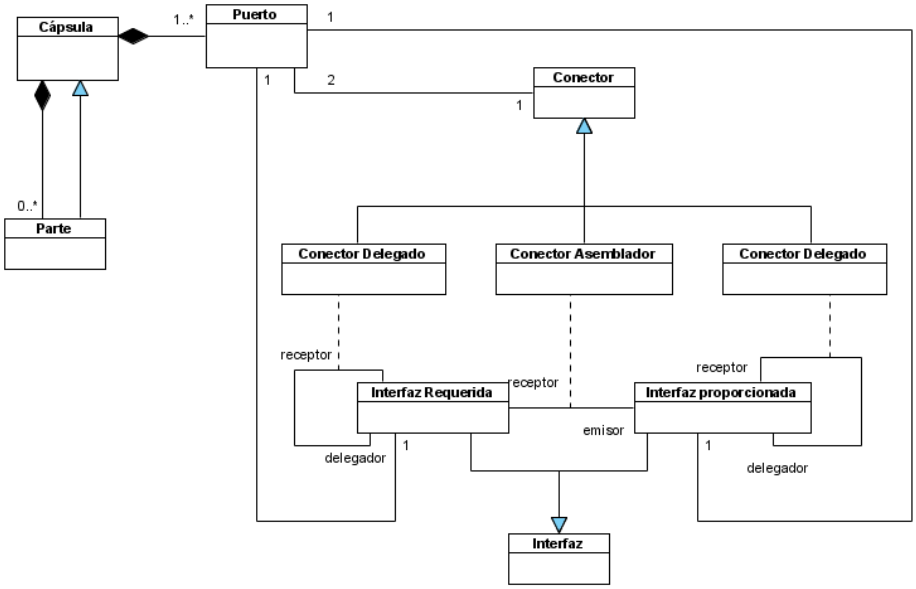


Figura 4.4: Metamodelo del diagrama de estructura compuesta

Conceptos	**Definiciones**
Cápsula	Es una parte modular del sistema, que puede ser considerada como una unidad que funciona de manera autónoma dentro del sistema.
Interfaz	Puntos de acceso que controlan el comportamiento de la cápsula.
Puerto	Son puntos de interacción sobre los cuales se exponen potencialmente las interfaces.
Conector	Representa un enlace a través del cuál las cápsulas comunican.

Cápsula: una cápsula en MEDISTAM–RT está compuesta de dos partes: interfaces y contenidos.

- El **contenido**: incluye subcápsulas que a su vez pueden ser clasificadas en dos tipos:

- *Compuestos:* cuando el contenido incluye otros subcápsulas.
- *Primitivos:* cuando el contenido no incluye ningún subcápsula.

- **Interfaces**: representan puntos de acceso que controlan todo el comportamiento de la cápsula. A estos puntos de acceso, se denominan interfaces. UML 2.0 define dos tipos de interfaces, que también adopta la metodología MEDISTAM–RT.

  - *Interfaces proporcionadas:* describen el conjunto de servicios que la cápsula debe ofrecer a su entorno exterior. En otras palabras, estos interfaces representan el conjunto de servicios que el entorno puede demandar de la cápsula.
  - *Interfaces requeridas:* especifica el conjunto de servicios que debe proveer el entorno a la cápsula para que este último pueda funcionar correctamente. Estas interfaces describen los servicios que deben proporcionar a la cápsula para que pueda funcionar conforme a su especificación.

Puertos: los puertos representan puntos de interacción entre una cápsula y su entorno, especifican las comunicaciones con las interfaces tanto requeridas como proporcionadas. Una característica importante de estos elementos se manifiesta en el hecho de que permiten proporcionar una interfaz (un servicio) a través de la frontera de la clase cápsula que lo contiene sin revelar la estructura y el comportamiento internos del mismo. Asimismo, se oculta la información acerca del integrante de la cápsula que se ha encargado de proporcionar (o requerir) un servicio al (o del) cliente.

Este mecanismo permite que una cápsula software de manera aislada cuente con toda la especificación, no solo interna, sino también la que se refiere a la especificación de los requerimientos para la integración de dicha cápsula con otras cápsulas software. La figura 4.5 representa una cápsula Pda con un puerto. A su vez, este puerto cuenta con dos interfaces: una interfaz requerida (IUSBIn) y una interfaz proporcionada (ISBOut).

Una manera de caracterizar los puertos, es hacerlo a través de su localización en el componente:

- *Un puerto relay*: Reside en el borde estructural del componente, por lo cual tiene dos lados. Un lado conecta el componente de su entorno y el

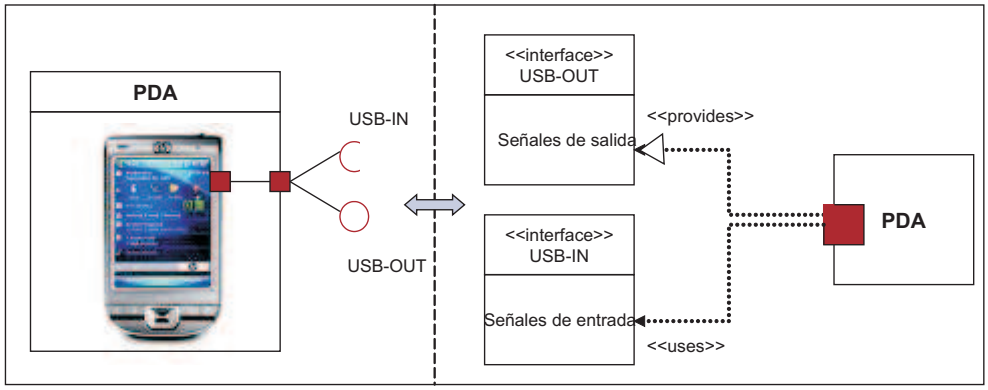


Figura 4.5: Vista de puertos de un PDA

otro lado conecta el componente a sus parte, asimismo transmite señales entre el componente y sus partes.

- *Un puerto end*: es un elemento que representa las señales entre el componente y sus diagramas de estados describiendo el comportamiento de las interacciones del componente.

Conector: un conector es el medio por lo cual interaccionan dos componentes. Asimismo, un conector representa un enlace, o medio de conexión, que da soporte hardware a las comunicaciones establecidas entre dos componentes.

Se definen dos tipos de conectores:

- *Conector Ensamblador:* es un conector que une una interfaz requerida (o puerto) a una interfaz proporcionada por otro componente. Ello indica que un componente provee los servicios requeridos por otro componente.

- *Conector Delegado:* Es un conector que une un componente con una de sus partes, de manera que los servicios requeridos o proporcionados al componente pueden ser delegadas a sus partes. Este tipo de conectores unen dos interfaces del mismo tipo, de tal forma que es posible transferir las llamadas invocadas desde su entorno a los sub-componentes y viceversa.

En definitiva, un conector es una abstracción del canal de transmisión de mensajes que conecta uno o más puertos. Cada conector es tipado por un

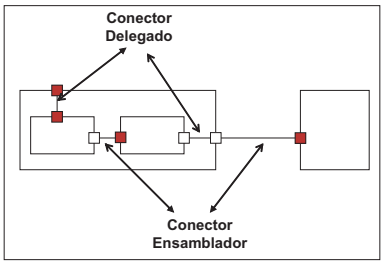


Figura 4.6: Vista de conectores

protocolo que define las posibles interacciones que pueden tener lugar a través de dicho conector.

4.3.2. Vista dinámica

La metodología que presentamos en esta tesis tiene como objetivo principal el diseño y el análisis de los STR. Las características de este tipo de sistema hace imprescindible contar con un lenguaje dotado de una buena capacidad expresiva para el modelado del aspecto dinámico del sistema, incluyendo el comportamiento temporizado del mismo. UML y UML–RT ha demostrado ser un buen método de diseño, pero sigue manifestando debilidad en la descripción del aspecto dinámico de los sistemas, sobre todo los STR, que son complejos, con tareas concurrentes que necesitan un especial tratamiento para asegurar la coordinación y la sincronización de estos. Especificar el tiempo de ejecución de los eventos que llevan a cabo estas tareas es de suma importancia para planificar el buen funcionamiento de los STR. Por esta razón, pretendemos en esta sección enriquecer la sintaxis de los diagramas *comportamentales* de UML, con nuevas construcciones para adaptarlo al modelado de los STR.

4.3.2.1. Diagrama de estados temporizado

Los diagramas de estadod de UML son herramientas gráficas que comprenden un conjunto extensivo de construcciones con una buena semántica estructural que permite la descripción del comportamiento del sistema. Para extender el alcance de estos sistemas de forma que puedan especificar adecuadamente el comportamiento temporizado de los STR, enriquecimos su sintaxis con nuevas

anotaciones inspiradas de la sintaxis del lenguaje CSP+T, conocido por su flexibilidad en la descripción del aspecto dinámico de los sistemas, así como las propiedades temporales del mismo.

Llamamos a esta extensión Diagrama de estados Temporizado. En la figura **??**, presentamos el metamodelo correspondiente al diagrama de estados temporizado para visualizar la extensión, en ello, vemos que una transición esta ahora compuesta de acciones y eventos temporizados del CSP+T, de esta manera hemos permitido:

1. Capturar el tiempo de ocurrencia de un evento. Cada transición etiquetada con el un evento marcador e_1, se transforma a una transición etiquetada por el mismo evento seguido por una acción $t = gettime()$ para capturar el tiempo de ocurrencia del evento y guardarla en la variable t (véase el cuadro 4.1(1)). Esta transformación apunta a la introducción de la noción del evento temporizado en UML–RT.

2. Restringir el tiempo de ocurrencia de un evento a un intervalo de tiempo absoluto (véase el cuadro 4.1(2)).

3. Restringir el tiempo de ocurrencia de un evento a un intervalo de tiempo relativo (véase el cuadro 4.1(3). Por ejemplo, en el cuadro 4.1(4), la ocurrencia del evento e_2 está restringida a un tiempo T después de la ocurrencia del evento e_1.

4. Disparar un evento especial llamado $Timeout$ que lleva el estado del componente a un estado $SKIP$, cuando un evento restringido no ocurre dentro del intervalo de tiempo especificado (véase el cuadro 4.1(4). Con esta extensión, pretendemos presentar restricciones temporales mediante la noción de eventos marcadores, restringidos y intervalos de habilitación. Para dos eventos marcadores sucesivos ($e1$, $e2$), asignamos al evento precedente $e1$ una variable marcadora $t1$ para registrar el tiempo de ocurrencia del evento ($e1 = gettime()$) y a e_2 (el evento sucesor) se le asigna un intervalo $I(T, t_1)$ en forma de guarda, $e_2[I(T, t_1)]$, un conjunto de reglas de transformación, siendo T un tiempo suficiente para la ocurrencia de e_2. Eso significa que la ocurrencia del evento $e2$ esta restringido a un tiempo T después de la ocurrencia del evento e_1; de otro modo, si

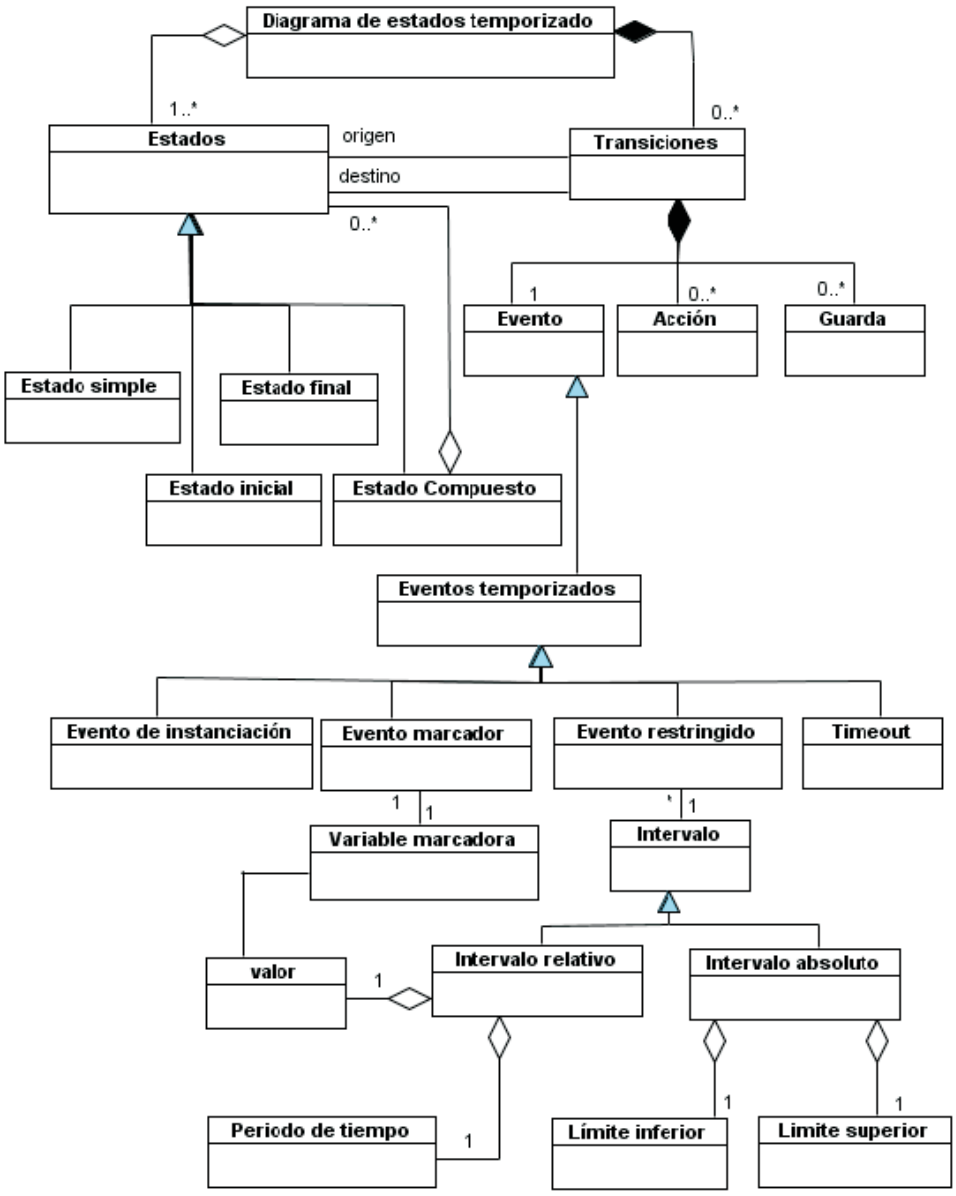
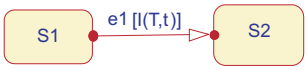
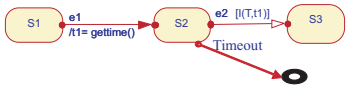


Figura 4.7: Metamodelo del diagrama de estados temporizado

el evento no ocurre durante el intervalo de *habilitación* I, se activa el evento especial $Timeout$ llevando el sistema a un estado $SKIP$.

Cuadro 4.1: Reglas de extensión

	transiciones	transiciones temporizadas
1.		
2.		
3.		
4.		

4.3.2.2. Diagrama de secuencia enriquecida con tiempo:

Los diagramas de secuencia muestran como los mensajes se intercambian entre objetos u otros clasificadores del sistema para llevar a cabo una tarea. Estos diagramas se pueden aplicar a diferentes niveles de detalle. Asimismo, los diagramas de secuencia se pueden usar a nivel de diseño detallado donde se tiene que establecer una comunicación entre cápsulas conforme a protocolos predefinidos.

Por un lado, como se ha mencionado antes, la interacción de cápsulas con el exterior se hace a través interfaces expuestos mediante puertos que representan la parte controladora del componente. Eso significa que los puertos son elementos activos que tienen un comportamiento propio. Además de esto, cada componente tiene diferentes puertos independientes que pueden funcionar de manera paralela. O sea, la visión de comunicación de un componente con su entorno se orienta hacia una visión de interacciones de cada de sus puertos con

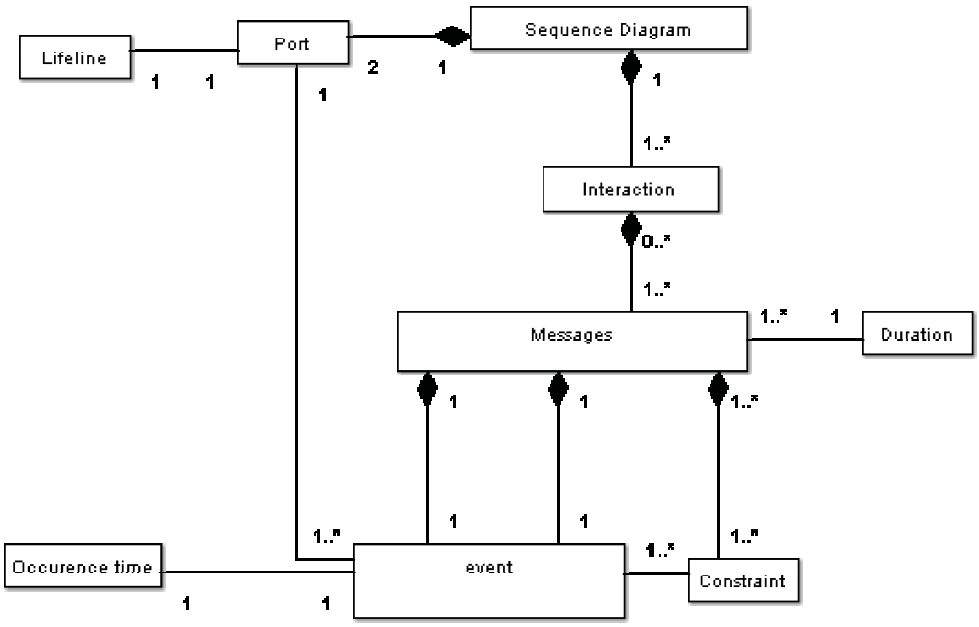


Figura 4.8: Metamodelo de TSD

el exterior. Por eso, en Mesdistam–RT los diagramas de secuencia, se utilizan de manera especial que satisface la vista de los procesos activos y que permita utilizar el principio *handshaking* del CSP.

Por otro lado, para que esta comunicación sea segura y correcta es imprescindible tomar en cuenta aspectos temporales que llevan a especificar restricciones temporales y que se asignan al comportamiento de las cápsulas. Por eso, se enriquece el diagrama de secuencia de UML, con anotaciones temporales, que permiten especificar restricciones temporales asignadas a las interacciones entre cápsulas.

Llamamos a la extensión del diagrama de secuencia, diagramas de secuencias temporizados (TSD). La figura 4.8, representa el metamodelo que visualiza está extensión. Los diagramas de secuencia temporizados representan el comportamiento externo de los componentes. Es decir las interacciones con el exterior. Estas interacciones están basadas en mensajes. Cada mensaje tiene asociado :

1. Un puerto emisor.

2. Un puerto receptor.
3. Un evento de envío que representa la acción de envío del mensaje por el emisor.
4. Un evento de recepción que representa la acción de la recepción del mensaje por el receptor.
5. Duración. El tiempo que pasa desde el envió hasta la recepción del mensaje.
6. Restricciones:
   a) Orden parcial entre eventos.
   b) Restricciones temporales.

4.4. Transformación de los modelos de MEDISTAM–RT al plano formal

Como ya se ha explicado antes, en la sección 4.2.1.1, con objeto de poder analizar y verificar un sistema modelado siguiendo el enfoque metodológico de MEDISTAM–RT, se ha definido para cada uno de los modelos propuestos en la metodología, un conjunto de reglas de transformación al lenguaje CSP+T. A continuación vamos a presentar cada uno de ellos.

4.4.1. Formalización dinámica de los diagramas de clase a procesos CSP+T

Definición 1: definimos las funciones:

- $Puertos : C_D \rightarrow \mathbb{P}_{\mathbb{D}}$ que devuelve el conjunto de puertos de una cápsula $C \in C_D$
- $Prot : P_D \rightarrow PT_D$ que devuelve el protocolo asignado a un puerto $P \in P_D$

- $Role : P_D \rightarrow Basico, Conjugado$ que devuelve el role de un puerto $P \in P_D$

Regla 1 (Cápsula): una *cápsula* representa un componente de un sistema, activo y auto contenido [SR98]. Esta misma definición coincide con la de *proceso* en el marco conceptual del CSP y CSP+T [Sch00]. Por lo tanto, la correspondencia de una cápsula UML a un proceso CSP es inmediata.

Para ello, definimos la función $T : C_D \rightarrow CSP+T$ que mapea cada cápsula en un proceso CSP+T, con la siguiente restricción[2]

$\forall C \in C_D nombre(T(C)) = C$. Al describir un proceso, lo primero que hay que definir es su alfabeto de comunicación, es decir el conjunto[3] de eventos que el proceso pueda realizar mediante interacciones con su entorno[4].

Definición 2(Alfabeto de comunicación): sea C una cápsula con n puertos. Los mensajes que envía o recibe una cápsula pasan a través de los puertos de la cápsula C. De ahí, estos mensajes deben estar encapsulados en los protocolos asignados a estos puertos. Formalmente,

$$x \in \alpha C \Rightarrow \exists P \in Puertos(C); x \in \alpha(Prot(P))$$

Deducimos que,

$$\alpha(C) = \alpha(Prot(Puerto_0)) \cup \alpha(Prot(Puerto_1)) \ldots \alpha(Prot(Puerto_n))$$

Entonces, el alfabeto de comunicación de un proceso CSP representando una cápsula C de UML es la unión de todos los alfabetos de comunicación de los protocolos asignados a sus puertos.

Regla 2 (Protocolo): cada protocolo tiene un comportamiento, lo que se puede describir como un proceso en CSP+T. De ahí, definimos la función $T_P : PT_D \rightarrow CSP + T$ que mapea cada protocolo en PT_D a un proceso en CSP+T con el mismo nombre (sea $PT \in PT_D$, $Nombre(T_P(PT)) = PT$).

El comportamiento de un protocolo PT se puede representar con el diagramo de estado temporizado de un componente C en un puerto p, anotado

[2]Esta restricción obliga llevar el nombre de la cápsula en su proceso de transformación.
[3]Este conjunto representa el marco de descripción de la cápsula
[4]El entorno pueden ser otros componentes del mismo sistema.

con $D_{C,P}(I)$ (ver la página 99, tal que, $prot(p) = PT \ \wedge \ role(p) = basico$, de ello:

$$T_p(prot) = CSP + T(D_{C,p}(I))$$

$$= CSP + T(D_C(I))/(Ev_e \cup A_e)$$

Regla 3 (Asociación entre dos cápsulas): dos cápsulas A y B pueden establecer una comunicación si: $\exists p_1 \in puertos(A)$ y $p_2 \in puertos(B)$, tal que,

$$Prot(p_1) = Prot(P_2) \wedge role(P_1) \neq role(p_2)$$

Una asociación entre dos cápsulas muestra la existencia de una comunicación y colaboración entre las cápsulas asociadas. Esto significa que las dos cápsulas deben sincronizarse en los elementos de intersección de sus alfabetos de comunicación. De ahí, La asociación entre dos cápsulas $CapsA$ y $CapsB$ se representa como una composición paralela de los dos procesos CSP+T correspondientes a las cápsulas asociadas.

Demostración
Sea Q el proceso CSP+T que representa el comportamiento de la asociación de dos cápsulas A y B.

$$(e,t) \in F_{TT}(Q) \Rightarrow ((e,t) \in (\alpha A - \alpha B) \wedge e \in F_{TT}(A) \wedge e \notin F_{TT}(B))$$

$$\vee (e \in (\alpha A - \alpha B) \wedge (e,t) \in F_{TT}(A) \wedge e \notin F_{TT}(B))$$

$$\vee (e \in (\alpha(a) \cap \alpha B) \wedge (t \in I_{e,A} \cap I_{e,A}) \wedge (e,t) \in F_{TT}(A)$$

$$\wedge e \in F_{TT}(B))$$

$$\Rightarrow (e \in F_{TT}(Q) \upharpoonright \alpha(A) \Rightarrow (e,t) \in F_{TT}s(A))$$

$$\vee (e \in F_{TT}(Q) \upharpoonright \alpha(B) \Rightarrow (e,t) \in F_{TT}s(B))$$

$$\vee (e,t) \in traza(Q) \upharpoonright (\alpha A \cap \alpha B) \Rightarrow t \in I_{e,A} \cap I_{e,A})$$

$$\Rightarrow (e,t) \in F_{TT}(A \underset{(\alpha(A) \cap \alpha(B)}{\parallel} B).$$

Regla 4 (Asociación entre más de dos cápsulas): sea C_1, C_2 y C_3 tres cápsulas conectadas de la siguiente manera: C_1 está conectada con la cápsula C_2 a través del protocolo pr_1 y C_2 conectada con la cápsula C_3 mediante el protocolo pr_2. Entonces C_2 debe sincronizar en $\alpha(pro_1)$ y $\alpha(pro_2)$ con los procesos C_1 y C_3, respectivamente.

llamamos Q, el comportamiento de las tres cápsulas C_1, C_2 y C_3 en conjunto.

y sea $Q_1, Q_2 \in C$ tal que $Q_1 = C_1 \underset{\alpha(pr_1)}{\|} C_2$ y $Q_2 = C_2 \underset{\alpha(pr_2)}{\|} C_3$ De ello,

$$Q = Q_1 \underset{\alpha(pr_1)}{\|} C_3 = Q_2 \underset{\alpha(pr_2)}{\|} C_1$$

Regla 5 (Puertos): los puertos se modelan con canales, eso se hace mediante un *renombramiento* del puerto al nombre del conector a la que se asocia.

El renombramiento esta establecido aquí para sincronizar las comunicaciones entre los procesos que representan las subcápsulas que comunican a través de estos puertos. De otra manera, si se utilizarían los nombres de los puertos asociados a los conectores, no se podría establecer ninguna comunicación entre los procesos del modelo CSP, porque cada proceso comunicaría a través de un canal diferente, Por ejemplo, La $CapsA$ en la figura 4.4.1 comunicaría solo a través del canal P_{a1} y la $CapsB$ a través del canal Pb, lo que llevaría a una situación de *interbloqueo*. Por eso, se renombran los puertos P_{a1} y P_b con el nombre del protocolo que les asocia ($Ra = [P_{a1} \rightarrow Ep1, P_b \rightarrow Ep1]$).

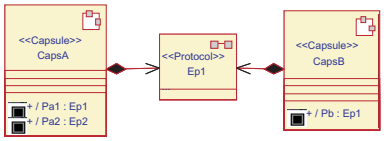


Figura 4.9: Asociación entre dos cápsulas

4.4.2. Transformación del diagrama de estructura compuesta a CSP+T

Los diagramas de estructura compuesta, aplican un *zoom-in* en la cápsula de manera que describe la arquitectura interna del sistema, es decir, los subcomponentes de la cápsula y cómo están conectados. Con la transformación de

este diagrama a procesos CSP, obtenemos una descripción de la arquitectura de los componentes en términos de procesos y operaciones CSP.
La transformación de los diagramas de estructura compuesta a términos de procesos CSP que presentamos aquí, originan del trabajo propuesto por Fischer en [FOW01].

Definición 3 un diagrama de estructura compuesta describiendo la estructura interna de una cápsula C, se representa con un tuple $EI = (C_D, P_D, Con)$, que consiste en:

- C_D: el conjunto de cápsulas en EI.
- P_D: el conjunto de puertos en EI.
- Con: el conjunto de conectores en EI, con $Con \subseteq P_D \times PT_D \times P_D$ (PT_D es el conjunto de protocolos en M).

4.4.3. Transformación dinámica del diagrama de estructura compuesta a CSP+T

Regla 6 (Conector): el conector representa el mismo concepto de protocolo, pero se diferencian en sus representaciones gráficas. Por lo tanto, sobre el mismo razonamiento sobre lo cual se basa la transformación del protocolo a un proceso CSP+T, se puede basar la transformación de un conector a un proceso CSP+T. Para ello, definimos una función $T_{con} : Con \rightarrow proceso\, CSP + T$ que mapea cada conector $c \in Con$ a un proceso CSP+T (con el mismo nombre del conector).

$$\forall\ c(p_1, pr, p_2) \in Con\ \Rightarrow \begin{cases} \alpha(T_{con}(c)) & = & \alpha(T_P(pr)) \\ T_{con}(c) & = & T_P(pr) \end{cases}$$

Regla 7 (cápsulas conectadas): subcápsulas dentro de una cápsula se ponen en composición paralela con una sincronización apropiada, tal y como esta definido por los conectores que relacionan las subcápsulas. Por ejemplo en la

figura 4.4.3, la conexión de las dos subcápsulas $CapsA$ y $CapsB$ se representan con :

$$CapsA \underset{C}{||} CapsB$$

-Los puertos siguen la misma transformación de los puertos en diagramas de clase.

$$Ra = Pa \rightarrow C1 \;\; Y \;\; Rb = Pb \rightarrow C1$$

Con el renombramiento, el mecanismo del *handshaking* se ve reflejado en la especificación de la comunicación entre las cápsulas compuestas en paralelo.

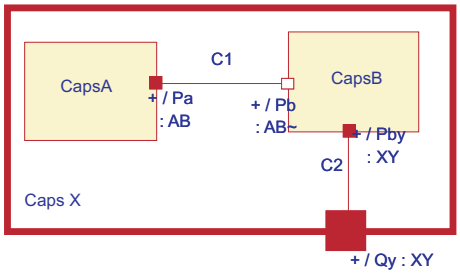


Figura 4.10: Diagrama de estructura compuesta de $CapsX$

Regla 8 (cápsulas compuestas en función de sus componentes): el proceso representando una cápsula compuesta se obtiene abstrayendo las comunicaciones internas de la cápsula, de manera que solo las comunicaciones con su entorno se quedan visibles. Para eso, se aplican dos mecanismos:

- Ocultamiento de los canales internos, por ejemplo en el caso de la $CapsX$ en la figura 4.4.3 hay que ocultar el conector $C1$ para que no transfiere ninguna comunicación desde o al exterior de la cápsula compuesta $CapsX$.

$$CapsX = \{CapsA||CapsB\} \setminus C1$$

- Renombramiento de los conectores exteriores al nombre del puerto publico en el borde de la cápsula compuesta. ya que estos conectores exteriores son conectores delegados que en sus extremos (puertos asociados) pasan las mismas comunicación. Por ejemplo, el canal $C2$ en la figura 4.4.3, es un canal exterior que transporta las comunicación desde el exterior a la subcápsula B y desde la subcápsula B al exterior, por lo tanto este

canal se renombra con el nombre del puerto publico externo Qy.

$$Rc = [C2 \rightarrow Qy]$$

Aplicando todas estas reglas de derivación se obtiene una especificación completa de la arquitectura interna de las cápsulas (componentes) compuestas. La $Capx$ en la figura 4.4.3, se representa mediante el proceso:

$$CapsX = (CapsA[Ra]||CapsB[Rb])[Rc]\backslash C1$$

con:

$$Ra = Pa \rightarrow C1$$

$$Rb = Pb \rightarrow C1, Pby \rightarrow C2$$

4.4.4. Transformación de los diagramas de estados temporizados

Definición 4 (Diagrama de estado temporizado de una cápsula): un diagrama de estado temporizado describiendo el comportamiento de un cápsula C, se representa con un tuple $M = (E_M, Ev_M, B, A_M, T_M, P)$, que consiste en:

- E_M: es un conjunto de estados en el diagrama M. Que a su vez esta compuesto de 4 conjuntos, E_{Ms}: conjunto de estados simples, E_{MC} conjunto de estados compuestos, E_{Mi} conjunto de estados iniciales, E_Mf conjunto de estados finales. A su vez, el conjunto E_{MC} esta compuesto con un conjunto, E_{Msn}, que representa el conjunto de estados que no están encapsulados.

- Ev_M: es un conjunto de eventos encontrados en el diagrama de estados temporizados de C. Estos eventos representan los eventos exteriores que recibe la cápsula C desde un puerto $p \in P$.

- B : es un conjunto de boléanos en el diagrama diagrama de estados temporizados de C.

- A_M :es un conjunto de acciones en el diagrama diagrama de estados temporizados de C. Estas acciones representan los eventos enviados por la cápsula C al exterior desde un puerto $p \in P$.

- T_M : es el conjunto de transiciones en el diagrama M. con:

$$T_M \subseteq E_M \times (Ev_M) \times (B \cup \{\ \}) \times (\bigcup A_M \cup \{\ \}) \times E_M$$

donde , $(s1, e, b, (a_1, a_2), s2) \in T_M$, lo cual $s1, s2 \in E_M, e \in E_V$, $b \in B$ o es nulo, $a \in A_M$ o es nulo.

- P es el conjunto de puertos de la cápsula C.

Definición 5: sea $n \in \Re$, definimos las funciones:

- *origen:* $T_M \longmapsto E_M$, que devuelve el estado origen de una transición.
- *extremo:* $T_M \longmapsto E_M$ que devuelve el estado extremo de una transición.
- *evento:* $T_M \longmapsto Ev_M$ que devuelve el evento disparador de una transición.
- *intervalo:* $Ev_M \longmapsto \Re \times \Re$ que devuelve el intervalo de habilitación de un evento.
- *acción:* $T_M \longmapsto (A_M * A_M)^n$ que devuelve la secuencia de acciones producidas en una transición $t \in T_M$.
- *guarda:* $E_M \longmapsto B$ que devuelve la guarda de un evento $e \in E_M$.
- $puerto_{in}$: $E_M \longmapsto P$ que devuelve el puerto de entrada de un evento $e \in E_M$
- $puerto_{out}$: $A_M \longmapsto P$ que devuelve el puerto de salida de una acción $a \in A_M$
- *inicial:* $E_{Mc} \longmapsto E_{Mi}$ que devuelve el estado inicial de un estado compuesto $c \in E_{Mc}$
- *final:* $E_{Mc} \longmapsto E_{Mf}$ que devuelve el estado final de un estado compuesto $c \in h$
- *super:* $E_M \longmapsto E_{Mc}$ que devuelve el estado que directamente encapsula un estado $e_s \in E_{Mc}$, tal que

$$super(e_s) = \begin{cases} \emptyset & \text{si} \quad e_s \notin E_{Msn} \\ C & \text{si} \quad e_s \in E_{Msn} \end{cases}$$

Definición 6 (Diagrama de estado temporizado de un puerto): el comportamiento de una cápsula C, se puede describir en vista de uno de sus puertos $p \in P$ mediante un diagrama de estado temporizado del puerto. Definimos el conjunto Ev_e de los eventos de la cápsula C que no son recibidos a través de un puerto p, con:

$$Ev_{e,\neg p} = \{e \,|\, p_e = puerto(e) \Rightarrow p_e \neq p\}$$

Definimos el conjunto $A_{e,\neg p}$ de las acciones de la cápsula C que no son enviados a través de un puerto p, con:

$$A_{e,\neg p} = \{a | p_e = puerto(a) \Rightarrow p_e \neq p\}$$

Representamos el diagrama de estado temporizado del puerto P con el tuple $(M, P) = (E_{ES}, Ev_{(M,P)}, A_{(M,P)}, T_{(M,P)})$ que consiste en:

- E_{ES}: es un conjunto de estados .
- $Ev_{(C,P)}$: es el conjunto de eventos en el diagrama de estado temporizado del componente C en el puerto P, definido como

$$Ev_{(M,P)} = \{e | e \in Ev_M \ \& \ e \notin Ev_{e,\neg P} \ \}$$

- $A_{(C,P)}$: es el conjunto de acciones en el diagrama $D_{C,P}$, definido como

$$A_{(C,P)} = \{a | a \in A_M \ \& \ a \notin A_{e,\neg P} \ \}$$

- $T_{(C,P)}$: es el conjunto de transiciones en el diagrama $D_{C,P}$.

4.4.5. Formalización dinámica del diagrama de estado temporizado

Para formalizar el comportamiento temporizado de un objeto descrito con Diagrama de estado temporizado, definimos una función de transformación

$$CSP + T : DET \longmapsto Proceso\ CSP + T$$

que mapea cada diagrama de estado temporizado a un proceso $CSP + T$. Según la semántica de los diagramas de estado, un objeto modelado puede

tener un conjunto de estados pero sólo se puede encontrar en un solo estado a la vez. El comportamiento de un componente M a partir de que esté en un estado J se describe con un diagrama de estado temporizado que anotamos con $D_M(J)$. De ahí, el comportamiento completo de M se representa con un $D_M(I)$, siendo I el estado inicial del componente.

Definición 7 (Comportamiento de un puerto) anotamos con $D_{C,p}(I)$ el diagrama de estado temporizado de la cápsula C en el puerto p a partir del estado inicial $I \in E_M$.

La especificación del comportamiento de una cápsula en uno de sus puerto se puede deducir de la especificación del comportamiento de la cápsula ocultando los eventos y acciones que no pasan por el puerto en cuestión. Definimos este comportamiento con:

$$(D_{C,p}(I)) = (D_C(I)) \setminus (Ev_{e,\neg p} \cup \ A_{e,\neg p})$$

Regla 9 (Estados y eventos): cada evento $e \in Ev_M$ se transforma a un evento temporizado del CSP+T y cada estado $s \in E_M$ se transforma a un proceso CSP+T. Para ello, definimos las funciones:

$$T_{Ev} : Ev_M \longmapsto Eventos\ temporizados$$

$$T_{Es} : E_M \longmapsto Proceso\ CSP + T.$$

Regla 10 (Estado inicial): sea I, un estado inicial no encapsulado [5]($I \notin E_{Mc}$). I es asignado a un término que incluye la ocurrencia del evento de instanciación $\star$, que representa el origen de tiempo global en el cual los procesos del sistema pueden comenzar la ejecución con la ocurrencia de algún evento temporizado disparador.

$$CSP + T(D_M(I)) = t_i.\star \longmapsto I$$

[5]Un diagrama de estado temporizado consta con un solo estado inicial no encapsulado.

t_i representa el tiempo de activación del componente M.

Regla 1.3 (Transiciones con estados simples): cuando un componente M en un estado S_1 recibe un evento disparador $ev \in Ev_M$, primero guarda el tiempo de recepción del evento (ejecutando la acción t=gettime()) y luego, manda las acciones de la transición t para pasar a comportarse como el estado extremo de la transición.

Figura 4.11: Transiciones con estados simples:

$$CSP + T(D_M(S_1)) = ev_1 \bowtie t_1 \longmapsto D_M(extremo(t) = S_2)$$

Regla 1.4 (Transiciones con estados compuestos): cualquier transición de $S1$ a un estado de destino compuesto $S2$ provocada por un evento marcador e es transformada en el término CSP+T dado por:

$$CSP + T(D_M(S_1)) = e \bowtie v \rightarrow D_M(inicial(S_2))$$

Y cualquier transición con un estado de origen compuesto $S1$ a $S2$ provocada por un evento marcador e es transformada en el término CSP+T dado por:

$$CSP + T(D_M(final(S_1))) = e \bowtie v \rightarrow D_M(S_2)$$

Por ejemplo la transición en la figura 4.4.5, corresponde al término de procesos.

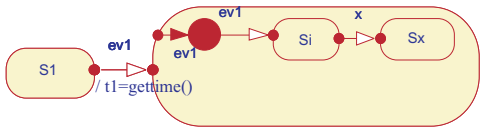


Figura 4.12: Transiciones con estados compuestos:

$$CSP + T(D_M(S_1)) = ev_1 \rightarrow D_M(S_i)$$

Regla 11 (Elección externa): la elección externa incluye a más de una transición saliente desde el estado origen. De ello, Representa una situación de

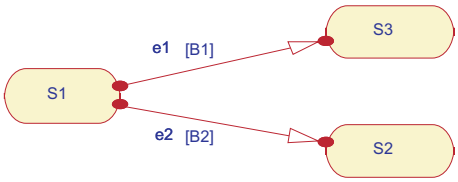


Figura 4.13: Transición externa con más de un estado destino

elección simple entre distintas formas de comportarse $S_2 \Box S_3$ según la elección del entorno,(La elección del primer evento visible).

Por ejemplo, en la figura 4.4.5 el objeto se puede comportar como $(e_1 \& B_1 \rightarrow S_2$, si el evento e_1 ocurre con la guarda $B_1 = true$, o pasa a comportarse como $e_1 \rightarrow S_3$ si el e_2 es el evento que se dispara desde el estado de origen S_1, siendo $B_2 = true$. o sea,

$$CSP + T(D_M(S_1)) = (\, e_1 \& B_1 \rightarrow D_M(S_1)\,) \Box (\, e_2 \& B_2 \rightarrow D_M(S_2)\,)$$

En general:

Sea J el estado de origen, por donde sale N transiciones t_i. El objeto en este estado se va a comportar como $D_M(J)$, Tal como:

$$CSP + T(D_M(J)) = \Box_{1..N}\, evento(t_i) \& guarda(evento(t_i)) \rightarrow D_M(extremo(t_i))$$

Regla 12 (Elección interna) UML representa una elección interna dibujando un diamante en las transiciones correspondientes, como se ve en la figura 4.4.5.

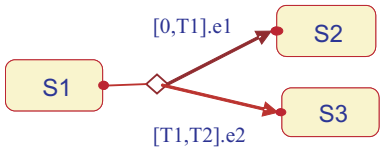


Figura 4.14: Transición interna con más de un estado destino

En este caso, la elección es interna del componente, es decir, no se le permite al entorno ningún control sobre la selección de los eventos. A cada evento está asignado un *intervalo de habilitación* que restringe su ejecución a un tiempo definido, de está forma se puede dar prioridad a eventos con respecto a otros. Por ejemplo, en la figura 4.4.5, se ha asignado el intervalo

$I[O, T_1]$ al evento e_1 para priorizar su ejecución frente al evento e_2 que se le ha asignado un intervalo $I[T_1, T_2]$. Asimismo, el componente se va a comportar como $D_M(S_2)$ si se da el evento e_1 dentro del intervalo $I[0, T_1]$ o como $D_M(S_2)$ si se da el evento e_2 dentro del intervalo $I[T_1, T_2]$, con $0 < T1 < T2$. Esto es,

$$CSP + T(D_M(S_1)) = (I[0, T_1].e_1 \rightarrow D_M(S_1)) \sqcap (I[T_1, T_2].e_2 \rightarrow D_M(S_2))$$

En general: sea J el estado de origen, por donde sale N transiciones t_i, el objeto en este estado se va a comportar como $D_M(J)$, tal que:

$$CSP{+}T(D_M(J)) = \sqcap_{1..N}(\, Intervalo(evento(t_i)).evento(t_i) \rightarrow D_M(extremo(t_i))\,)$$

Regla 12 (Elección condicionada): está elección se puede considerar como una especialización de la elección externa, donde, se sale dos transiciones t_1 y t_2 de un estado origen S_o. Si el componente acepta involucrarse en un evento e dentro de su intervalo de habilitación pasa a comportarse como $extremo(t_i)$, si se vence el intervalo de habilitación sin que el componente acepte el evento e, el proceso pasa a comportarse como el proceso $SKIP$.

$$CSP + T(D_M(S_o)) = Intervalo(e).e \ \Box \ Intervalo(e) \rightarrow SKIP$$

Por ejemplo en la figura 4.15,

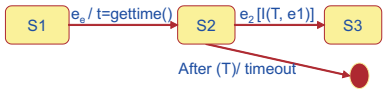


Figura 4.15: Timeout

$$CSP + T(D_M(S_1)) = e1 \bowtie t \rightarrow D_M(S_2)$$

$$D_M(S_2) = I(T, t).e_2 \rightarrow D_M(S_3) \ \Box \ I(T, t) \rightarrow SKIP$$

4.4.6. Transformación de los diagramas de secuencia temporizados

Definición 8: un diagrama de secuencia temporizada es el triple $SD = (P, M, TC)$, que consiste en:

- P representa un conjunto finito de *puertos*.
- M representa un conjunto finito de mensajes tal que, $M = (P_e, P_{re}, m, t_e, t_{re})$, con,
  - $P_e \in P$ es el puerto de envío del mensaje m
  - $P_{re} \in P$ es el puerto de recepción del mensaje m
  - t_e representa el tiempo del envío del mensaje m
  - t_{re} representa el tiempo de recepción del mensaje, con $t_e > t_{re}$
- TC es el conjunto de restricciones temporales de la forma $t_{re} - t_e^{'} \leq d$.

Definición 9: cada mensaje m consta de un evento de envío $!m$ y un evento de recepción $?m$. El envío del mensaje m en un tiempo t_e se representa con el término $(P_e.(!m) \bowtie t_e)$ y la recepción del mensaje en un tiempo t_{re} se representa con el término $(P_re.(?m) \bowtie t_{re})$.

Denotamos con N_e el conjunto de todos lo eventos enviados o recibidos en SD y $N_{e,p}$ el conjunto de eventos enviado o recibidos a través del puerto p. con TT anotamos el conjunto de trazas temporizadas(descrito en detalle en el capítulo 3).

Definimos las funciones:

- $tiempo : N_e \rightarrow \Re$, que devuelve el tiempo de ocurrencia (envío o recepción) de un evento $e \in N_e$

4.4.7. Formalización dinámica de DST

Cada diagrama de secuencia se va a representar como una traza temporizada, es decir una secuencia de eventos temporizados

$$\sigma_i = (e_0, tiempo(e_0))^\frown(e_1, tiempo(e_1))...^\frown(e_n, tiempo(e_n))$$

La representación gráfica de las interacciones básica lleva a cabo la significación intuitiva del orden parcial en la ocurrencia de los eventos. Definimos la función $TTraza : SD \rightarrow TT$ que mapea cada diagrama de secuencia temporizado a trazas temporizadas.

Regla 13 (Interacción básica): vamos a empezar con la transformación de una interacción básica, que cuenta con un solo mensaje (m_1) como se muestra en la figura 4.16. La interacción básica se va a especificar como la concate-

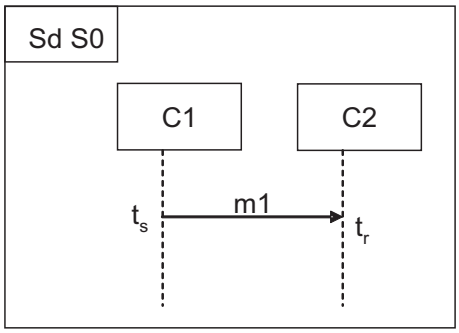


Figura 4.16: Mensaje simple

nación del evento temporizado de envío y evento temporizado de recepción del mensaje.

$$TTraza(SD) \;=\; \langle((!m_1, tiempo(!m_e)) \frown (?m_1, tiempo(?m_e)))\rangle)$$

Regla 14 (Secuencia de interacciones):una secuencia de dos interacciones se especifica con la concatenación de los mensajes sucesivos, m_p, m_s en orden de ocurrencia, sobre los cuales se basa las interacciones, con la condición de que el tiempo del envío del mensaje sucesor sea mayor que el tiempo de la recepción del mensaje predecesor. $tiempo(!m_s) > tiempo(?m_p)$.

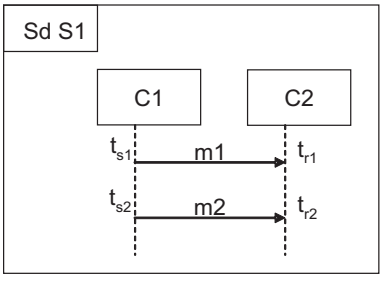


Figura 4.17: Secuencia de mensajes

$$TTraza(SD) = < m_p, tiempo(!m_p), tiempo(?m_p) >$$

$$\frown < m_s, tiempo(!m_s), tiempo(?m_s) >$$

Por ejemplo, el diagrama de secuencia S_1 en la figura 4.17 se especifica en términos de trazas de esa manera: $S1 =< m_1, t_{s1}, t_{r1} > \frown < m_2, t_{s2}, t_{r2} >$

Regla 15 (Referencia): una serie de interacciones se pueden combinar y referenciar para manejar la complejidad de la representación de las interacciones de manera que, en un diagrama de secuencia se puede representar un bloque etiquetado con la referencia de la combinación. Este bloque referenciado se trata como una interacción básica. Por ejemplo, en el diagrama de secuencia S_1 (en la figura 4.18), las interacciones del diagrama de secuencia S_0 en la figura 4.16, está representado con el bloque etiquetado Ref S_0.

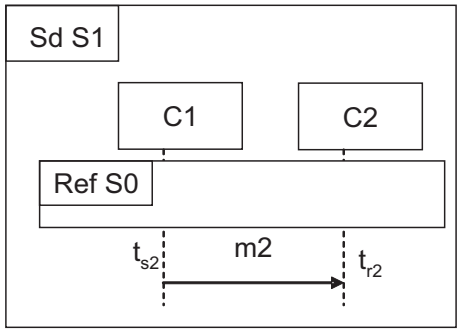


Figura 4.18: La etiqueta ref

$$S1 = S0 \frown m_2$$

$$=<!m1, ts1> \frown <?m1, tr1> \frown <!m2, ts2> \frown <?m2, tr2>$$

Regla 16 (Interacciones Alternativas) la figura 4.20, muestra como la ejecución condicional se presenta con una etiqueta *alt* y dividiendo el cuerpo

de control en múltiples sub-regiones con líneas discontinuas horizontales. Sólo el conjunto de mensajes de una sola región se puede ejecutar. Este diagrama de secuencia representa el comportamiento definido por la unión de la secuencia de trazas de ambas regiones. El diagrama de secuencia en la figura 4.20, se mapea a la secuencia de trazas siguiente:

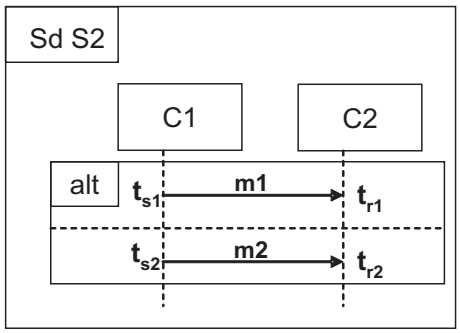


Figura 4.19: Ejecución paralela

$$S2 = m2|m1$$

$$=< m1, ts1, tr1 > | < m2, ts2, tr2 >$$

$$= (<!m1, ts1 > \frown <?m1, tr1 >)|(<!m2, ts2 > \frown <?m2tr2 >)$$

Regla 17 (Interacción Paralela) el operador "Par"debe ser interpretado como un o exclusivo. El cuerpo del operador de control esta divido en múltiples sub-regiones con líneas discontinuas. Cada sub-region representa una computación individual con eventos y acciones que se intercalan. Cuando se entra al operador de control, todas las sub-regiones se ejecutan concurrentemente. En este contexto, La concurrencia se debe interpretar como indistinguible de lo no-determinismo. La ejecución de los mensajes en cada región es secuencial, sin embargo el orden relativo de los mensajes en la subregiones paralelas son completamente arbitraria.La semántica de este operador se ilustra con el mapeo de la figura 4.20 a trazas temporizadas del CSP+T.

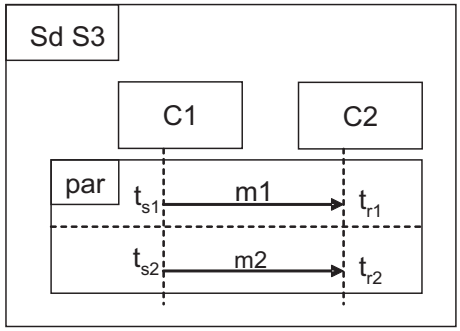


Figura 4.20: Ejecución paralela

$$S3 = (<!m1, ts1> \frown <?m1, tr1>) \frown (<!m2, ts2> \frown <?m2, tr2>)$$

$$\vee$$

$$(<!m2, ts2> \frown <?m2, tr2>) \frown (<!m1, ts1> \frown <?m1, tr1>)$$

Regla 18 (Interacción en Bucle) en la figura 4.21, se introduce un nuevo operador llamado ejecución iterativa representado por la etiqueta *loopn*. El cuerpo de control se ejecuta n veces repetidamente La semántica de este

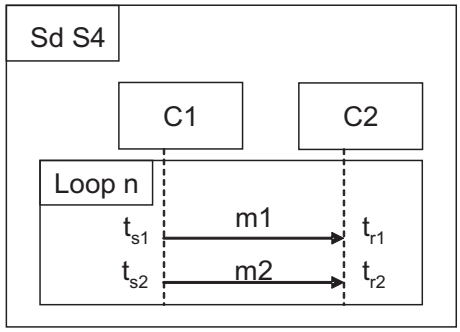


Figura 4.21: Ejecución en bucle

operador en CSP viene dada por:

$$S4 = (m1 \frown m2)^n$$

$$= (<!m1, ts1> \frown <?m1, tr1> \frown <!m2, ts2> \frown <?m2, tr2>)$$

Capítulo 5

MEDISTAM-RT

5.1. Vista general

MEDISTAM–RT (Método de Diseño de Sistemas basado en la Transformación Analítica de Modelos del Tiempo Real), tiene como propósito soportar al arquitecto de software en la especificación completa de un STR usando UML-RT y CSP+T. Como se puede observar en la figura 5.1, el enfoque propuesto está dividido en dos fases principales: una para modelar el sistema usando UML-RT y la otra para obtener la especificación formal según la sintaxis del álgebra de procesos en términos de CSP+T.

La transformación comienza a partir de un diagrama inicial de clases UML-RT, siguiendo una estrategia top-down, e integrando diagramas de estructura compuesta y diagrama de estados temporizados, con la finalidad de describir completamente el comportamiento de los componentes (cápsulas) del sistema y especificar formalmente las restricciones de tiempo real definidas como parte de los procesos del sistema.

Esta transformación se realiza dentro del marco de soporte conceptual proporcionado por los constructores notacionales de UML. De una forma más técnica, se puede afirmar que el enfoque metodológico propuesto es un nuevo

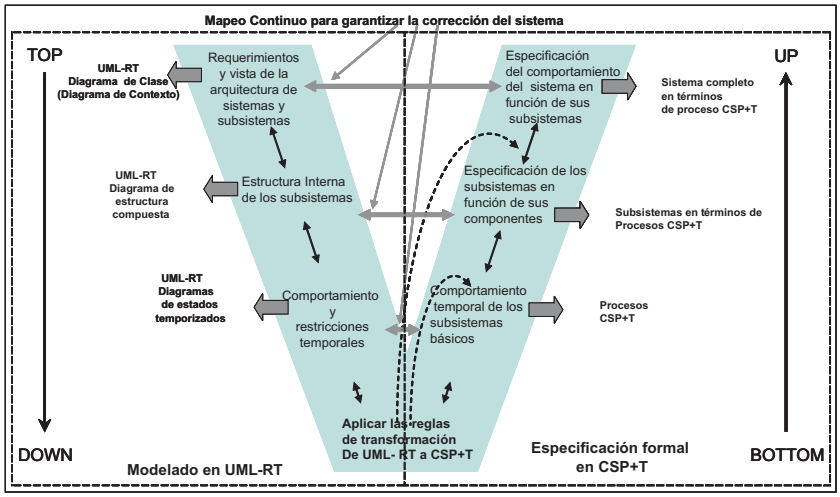


Figura 5.1: Vista general de MEDISTAM-RT

procedimiento de transformación sistemática para obtener el diseño completo de un STR, dándole una semántica operacional estructural, en términos del álgebra de procesos CSP+T, a las entidades semiformales de modelado de UML-RT. El resultado *completo*[1] es obtenido en la segunda fase, a través de una estrategia bottom-up. Como muestra la figura 5.1, a lo largo de todo el proceso existen vínculos de mapeo entre los diagramas UML-RT que modelan los componentes estructurales del sistema y su especificación formal *equivalente*[2] en términos de procesos CSP+T. Estos vínculos establecen cómo los términos sintácticos CSP+T se utilizan para representar restricciones de tiempo real, así como para representar los componentes internos y conectores que constituyen, a diferentes niveles de abstracción, una descripción detallada de la arquitectura del sistema, durante todo el proceso de transformación. Con la transformación de los diagramas UML-RT de los componentes del sistema en procesos CSP+T se realiza con la ayuda de un sistema de reglas, garantizando que el diseño final del sistema que se obtiene sea *"correcto por construcción"*.

[1] La especificación formal completa del sistema que se quiere diseñar.

[2] El conjunto de reglas de transformación que se proponen en el esquema transformacional permite afirmar dicha equivalencia.

5.2. Análisis de requisitos

Al igual que otras propuestas metodológicas de diseño de sistemas en general, MEDISTAM-RT comienza una etapa del análisis de los requerimientos; Un requerimiento representa necesidades que debe satisfacer, o condiciones que debe cumplir el sistema [CMC06]. El primer objetivo de esta etapa es determinar y documentar lo que se espera que haga el sistema. La metodología no incluye la tarea de elicitación de requisitos [LRA02][3], ya que no es una metodología del desarrollo sino de diseño y análisis.

MEDISTAM-RT recibe como entrada el documento generado en la tarea de elicitación. MEDISTAM-RT empieza en la segunda etapa de la ingeniería de los requisitos que es la fase de análisis de los requerimientos.

La información de entrada que requiere la metodología es:

- Los actores que interactúan con el sistema objeto de estudio. Estos actores pueden ser personas, dispositivos o cualquier otro elemento con quien debe el sistema comunicar.
- El conjunto de requerimientos que el sistema debe satisfacer.

Los requerimientos funcionales de un sistema especifican servicios que el sistema debe proporcionar cuando recibe las peticiones correspondientes del exterior. Estos requerimientos se satisfacen mediante una interacción entre el sistema y su entorno. La representación en los modelos MEDISTAM-RT recoge:

1. La información requerida por el sistema para llevar a cabo el servicio. Está información se representa como *entradas* del sistema.
2. La información proporcionada por el sistema a su entorno como resultado de la prestación del servicio. Esta información se representa como *salidas* del sistema.

Para modelar esta información, necesitamos un diagrama que muestra la lista de mensajes que representan las interacciones entre los actores y el sistema. Los casos de uso de UML carecen de mecanismos para capturar esta vista,

[3]Esta fase contempla las tarea de la obtención de información sobre el dominio del problema, identificación y revisión de objetivos del sistema, y finalmente la priorización de los objetivos y los requisitos del sistema [TJ]

por esta razón, diferentes trabajos han propuesto el uso de los diagramas de contexto utilizados en los métodos tradicionales estructurados para llenar el hueco que dejan los diagramas de caso de uso.

Bruce Duglass en su libro [Dou97], propugna el uso del diagrama de contexto de los métodos estructurados tradicionales directamente en el contexto del orientado a objetos, y sostiene su suposiciones observando que un diagrama de contexto muestra realmente objetos en lugar de procesos funcionales. Asimismo, indica la relevancia que puede tener la posibilidad de capturar las interacciones entre el sistema y sus actores en forma de mensajes y eventos en el modelado de los requerimientos.

En la aproximación de MEDISTAM-RT, utilizamos los diagramas de clase de UML-RT, en una forma especial y a diferentes niveles de abstracción, empleándolos como un tipo de diagramas de contexto.

Este diagrama mixto combinaría el alcance de los dos tipos de diagramas. De está manera, el diagrama de contexto puede ser utilizado para dar una vista unificada del contexto del sistema junto con su vista interna. Esto es, la interacciones del sistema con su entorno y las interacciones entre componentes del sistema que se modela mediante los protocolos.

Esta representación representa la ventaja de poder *aislar* partes del sistema, en el sentido de que permite especificar los requerimientos que debe cumplir cada uno de los componentes o subcomponentes según el nivel de abstracción, esto permitiría a cada componente ser diseñado, analizado y verificado *independientemente* de los otros componentes del sistema.

Así, representamos los elementos extraídos del documento de entrada de la metodología en *un diagrama de contexto inicial*, de alto nivel, donde el sistema completo es considerado como una caja negra y representado como una capsula. Como ejemplo de este considere el sistema *sistemacompleto* de la figura 5.2

Tanto el sistema, como los autores pueden enviarse y recibir mensajes mutuamente. Las señales intercambiadas entre la cápsula y cada actor son modelados con flechas en las mismas direcciones de las comunicaciones.

En general, los requerimientos de los subsistemas comunicantes están relacionados entre ellos, es decir, no se pueden capturar de manera independiente. Es necesario mostrar esta dependencia en el documento de análisis para:

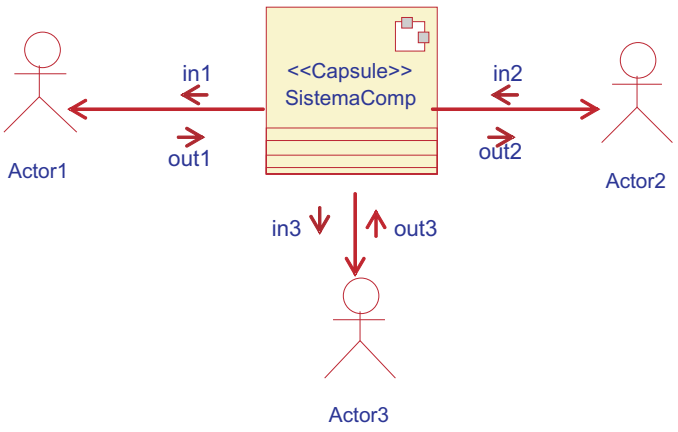


Figura 5.2: Diagrama de contexto del sistema completo

1. Mejorar la reusabilidad

2. Establecer la consistencia entre los requerimientos.

Adicionalmente, para los sistemas complejos, el análisis de los requerimientos y el diseño son actividades entrelazadas. En este sentido, un diseño arquitectónico del sistema es necesario para identificar los subsistemas y sus relaciones. Los requerimientos para estos subsistemas también deben ser especificados. Para ello, se utiliza una estrategia de *refinamiento paso a paso* que divide el proceso del modelado en diferentes etapas. Cada etapa consta de dos partes:

- **Requerimiento**: los objetivos que especifican lo que se debe cumplir en esta etapa.

- **Diseño**: la solución creada en esta etapa para satisfacer el requerimiento.

De esta manera, el diseño de una etapa del proceso se convierte en requerimientos para las etapas subsiguientes. Así, no se hace necesario en estas etapas volver atrás para ver de donde viene estos requerimientos. Esto representa una gran ventaja, ya que se reduce la dependencia y la comunicación entre las diferentes etapas del proceso ayudando a obtener un único y conciso conjunto de requerimientos de una etapa a otra a lo largo de la diferentes etapas.

5.3. Modelado del sistema

Con esta etapa se elabora el diseño completo de un sistema tiempo real utilizando tanto los diagramas de UML 2.0 y UML-RT como los diagramas extendidos de UML y expuestos en el capitulo 5. a continuación representamos las estrategias usadas a lo largo del proceso de modelado de sistemas.

5.3.1. Descomposición jerárquica del sistema

Dada la complejidad de los sistemas de tiempo real , la descomposición del sistema en módulos, o sistemas más pequeños y más abordables, se ha convertido a una etapa crucial de los métodos de diseño y análisis.

La composición representa una forma de refinamiento. En cada etapa de la descomposición jerárquica, se permite identificar la estructura interna del componente compuesto en cuestión (identificar los subcomponentes), e implícitamente, define la conexión entre los subcomponentes mencionados. De esa manera, estaríamos representado con más detalle el sistema en cada descomposición.

En MEDISTAM-RT, las interacciones entre los diferentes componentes y sub-componentes del sistema son considerados como base del proceso de la descomposición del sistema. El resultado de cada descomposición refleja implícitamente la solución propuesta por el diseñador para resolver el problema. Dado que en esta etapa el diseñador toma las decisiones de qué componentes deben colaborar para realizar el requisito, y también, delegar responsabilidades del sistema a componentes o subcomponentes específicos.

5.3.2. Descomposicion jerarquica de las tareas del sistema

La descomposición de las tareas es la base del modelo composicional; Los componentes en el modelo composicional están relacionados directamente con tareas en la composición de tareas [BDKTV99].

En realidad, cuando descomponemos el sistema en partes más pequeñas estamos sistemáticamente descomponiendo las tareas,que se requiere al sistema que proporcione, a tareas más pequeñas que llamaríamos subtareas. Cada

subtarea se delega a una parte especifica del sistema y representa el requerimiento que una parte del sistema debe cumplir.

Además, podemos decir que la necesidad de diseñar y obtener un comportamiento más detallado describiendo el sistema como una secuencia de tareas primitivas[4] conduce el proceso de la descomposición del sistema.

5.3.3. Transformación de modelos: abstracción y refinamiento

La abstracción y el refinamiento [HVF+05] [LRJ01] [LJ97] son dos transformaciones fundamentales y complementarios que se aplican durante el diseño de sistemas. La abstracción consiste en ocultar o eliminar información irrelevante, para mejorar la comprensibilidad del dominio del problema y simplificar la evaluación de la soluciones de diseño aportadas, facilitando de esa manera la toma de decisiones. Sin embargo, el refinamiento consiste en añadir más detalles al diseño, para acercarse a la solución final del sistema, eso es la implementación del mismo.

El objetivo mayor del refinamiento se manifiesta en reducir el hueco entre un modelo y su realización. Por ello, tiende a clarificar *como* se realiza la funcionalidad de un sistema o componente.

Precisamente, cada abstracción aplicada a un diseño responde de manera más clara a la pregunta ¿*Que* debe hacer el sistema diseñado? y cada refinamiento tiende a responder a la pregunta ¿*como* debe comportarse el sistema para alcanzar sus objetivos?

Una decisión importante es encontrar el nivel de abstracción más adecuado para la descripción de un componente o sub-componente. Es recomendado empezar con modelos los más abstractos posibles, pero sin demasiada perdida de información. El proceso de diseño en MEDISTAM-RT se puede considerar como una serie de pasos de abstracción/refinamiento, que parte de la especificación de las propiedades deseadas del sistema agregando detalles progresivamente hasta llegar a una implementación [AG97].

[4]Las tareas primitivas son tareas *descomponible* y las tareas compuestas son tareas que no se pueden descomponer

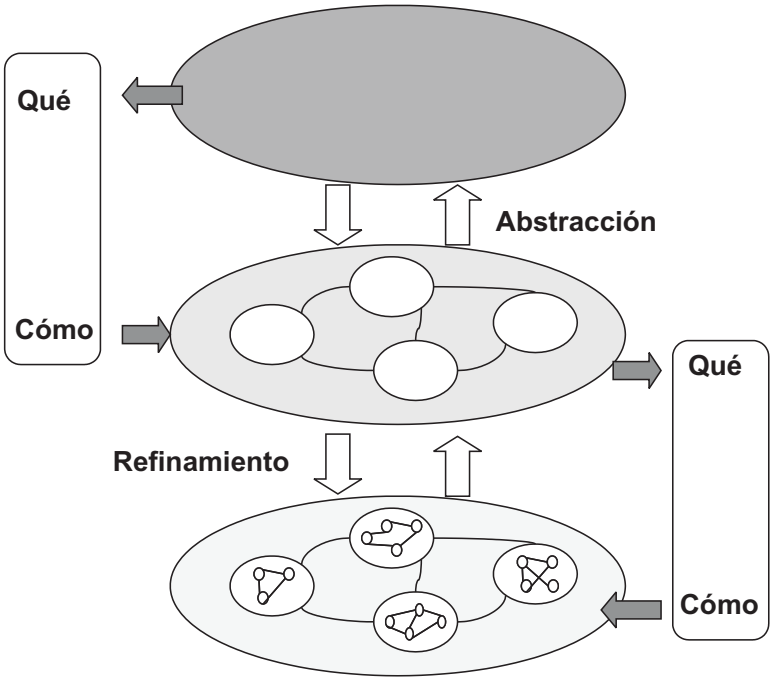


Figura 5.3: Transformación de modelos

5.3.4. Proceso de modelado top-down

El proceso comienza con una descomposición del sistema objetivo en un conjunto de subsistemas. Para luego, modelar cada subsistema de manera independiente. Cada subsistema puede a su vez ser descompuesta hasta llegar al nivel de refinamiento adecuado, esto es, un nivel donde todos los subsistemas son primitivos. Generalmente, se considera que para construir sistemas con la garantía de un nivel de calidad esperado y de una manera que sea eficaz respecto de la reducción de costes, es esencial construir un modelo global (arquitectura del sistema) en la fase de diseño del ciclo de desarrollo del sistema, que integre todos los aspectos del sistema. La arquitectura completa del sistema se obtiene mediante la combinación de las distintas vistas arquitectónicas [Kru95] que proporcionan los diagramas de clase (ilustran la arquitectura de los componentes de software y las dependencias entre ellos), los diagramas de estructura compuesta (capturan la estructura interna de cada componente)

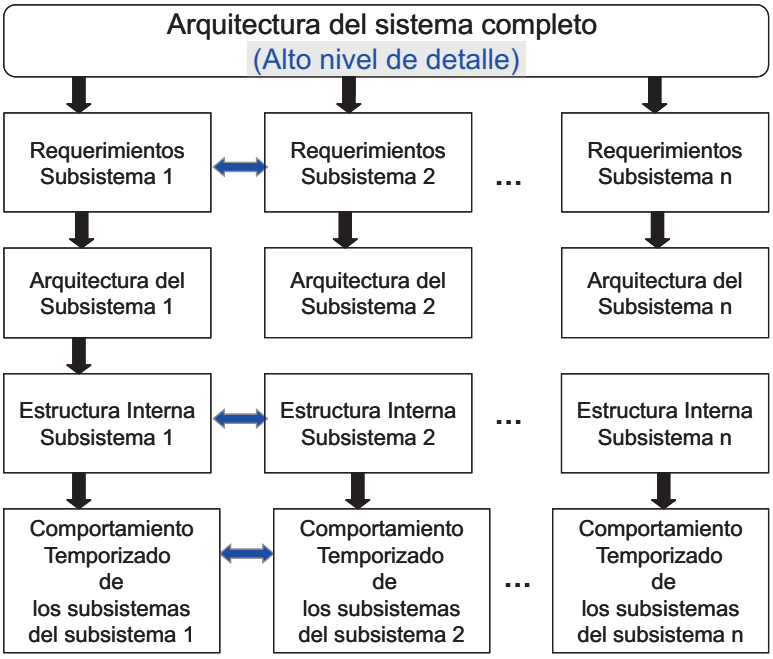


Figura 5.4: Estructura del proceso de modelado de sistemas

y los diagramas de estados temporizados (describen las interacciones a bajo nivel entre componentes). Los diagramas de transición de estado son muy importantes en la especificación de STR, ya que describen los aspectos de comportamiento y los cambios de estado de cada componente a lo largo del tiempo en un modelo STR.

5.3.4.1. Modelado de los requerimientos y de la arquitectura del *sistema completo*

La arquitectura del sistema completo se considera como una red de capsulas conectadas por puertos que se intercomunican de acuerdo con protocolos definidos previamente. Esta red se modela usando diagramas de clase, los cuales representan cada componente del sistema a través de cápsulas UML-RT y definen las interacciones a través del conjunto de operaciones (señales) encapsuladas en los protocolos.

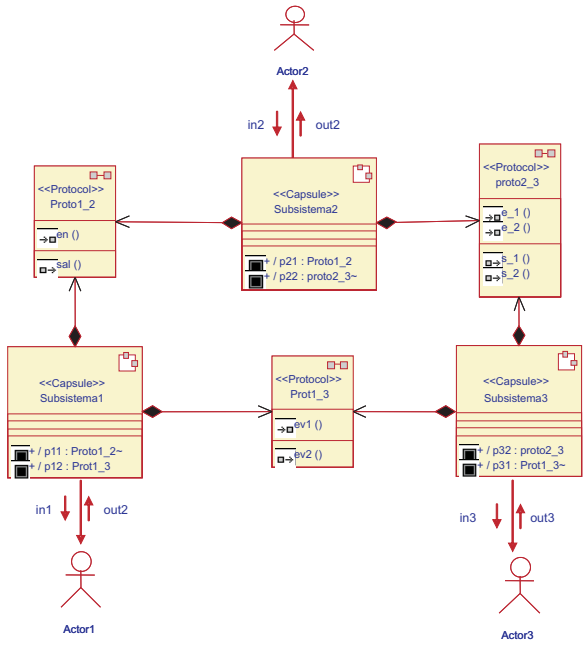


Figura 5.5: Estructura del proceso de modelado de sistemas

Los requerimientos del sistema completo modelado en el *diagrama de contexto inicial*, se asignan a los subsistemas resultantes de la descomposición del sistema. De está manera, se reparten las responsabilidades de prestación de servicios que debe proporcionar el sistema a los subsistemas.

Asimismo, se extiende el diagrama de clase elaborado para formar un diagrama de contexto en el nivel de abstracción de la descomposición del paso actual. La figura 5.5, muestra la arquitectura del sistema *sistema Completo*, el cual ha sido dividido en tres subsistemas y también muestra los requerimientos otorgados a cada subsistema.

5.3.4.2. Modelado de los requerimientos de cada subsistema

Al modelar un subsistema consideramos todo lo que es externo a el, como perteneciente al *entorno* del subsistema en cuestión. Asimismo, todos los subsistemas que interactúan con el subsistema en cuestión son considerados por este, sistemas externos tal y como muestra la figura 5.6.

Asimismo, los requerimientos de un subsistema se capturan en un diagrama

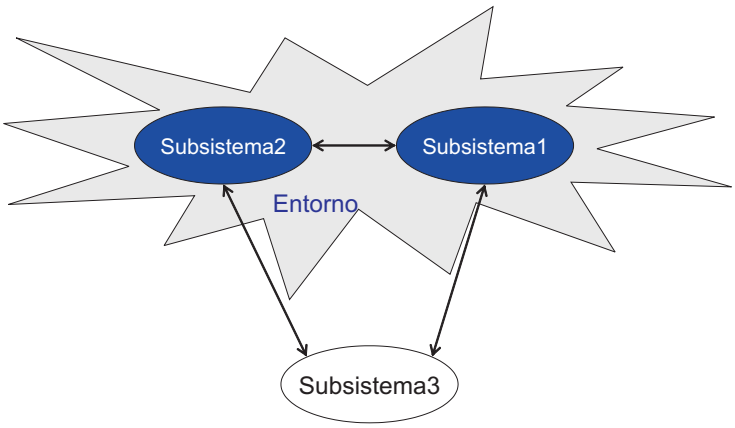


Figura 5.6: Entorno del *susbistema 1*

de contexto donde el subsistema es representado como caja negra, y sus actores son además de los actores externos al sistema y interactuando con el subsitema en cuestión, los subsistemas externos al subsistema y que interactúan con él. Por ejemplo, considérese, el *subsistema 1* en la figura 5.5, interacciona con el *Actor 1*, el *subsistema 2* y el *subsistema 3*. Asimismo, el *diagrama de contexto del sistema1* tendrá tres actores: *Actor 1*, el *subsistema 2* y el *subsistema 3*, tal y como se ve en la figura 5.7.

Además, las comunicaciones entre el actor y el subsistema se encapsulan en un protocolo de comunicación como se ve en la figura 5.8. Obsérvese que para cada subsistema x, se ha creado un puerto para comunicar con el actor x,($x \in \{1, 2, 3\}$). A Este puerto está asignado un rol básico del protocolo creado. Por ejemplo, en el *subsistema 1*, se ha creado un puerto $p_e : Actor1$, para guardar la información de la comunicaciones entre el *subsistema 1* y el *actor 1* dentro del subsistema.

En el protocolo común de los dos subsitemas (el subistema actor y el sistema objeto de modelar), se encapsulan los mensajes que se intercambian mutuamente. Cada subsistema tiene un puerto con un rol bien básico ,o bien conjugado, de ese protocolo común. Asimismo, las entradas y las salidas entre el subsistema y el actor se modelan de la siguiente manera:

- Si el subsistema objeto de modelado tiene el puerto básico, el sistema va a atender las peticiones encapsulados en los mensajes de entrada

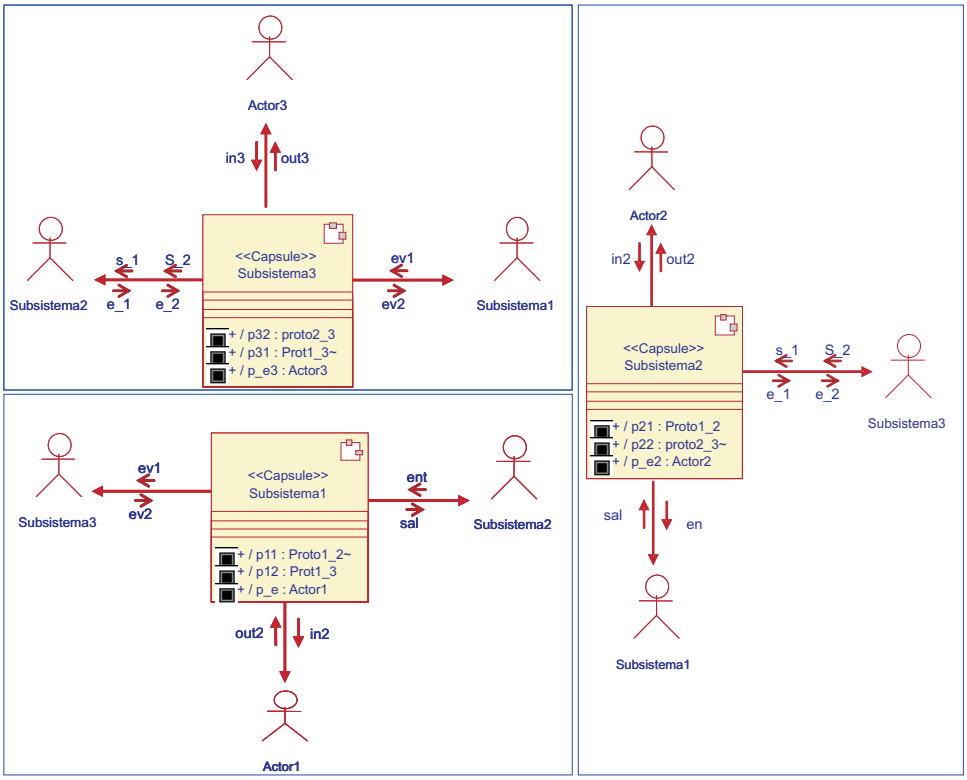


Figura 5.7: Diagramas de contexto de los subsistemas

del protocolo común, y proporcionar los servicios encapsulados en los mensajes de salida de este protocolo.

- En caso contrario, las peticiones del actor son las encapsuladas en los mensajes de salida y los servicios proporcionados son los encapsulados en los mensajes de entrada del protocolo.

5.3.4.3. Modelado de la arquitectura de los sub-sistemas

Después de dividir el subsistema en subsistemas de más bajo nivel, se modela su arquitectura con un diagrama de clase de UML-RT, que muestra los componentes que lo conforman y los señales que se intercambian. En la figura 5.3.4.5, se muestra la arquitectura del *subsistema 1*.

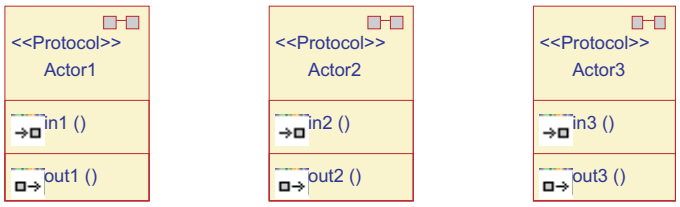


Figura 5.8: Protocolos de comunicación entre los actores y el *subsistema1*

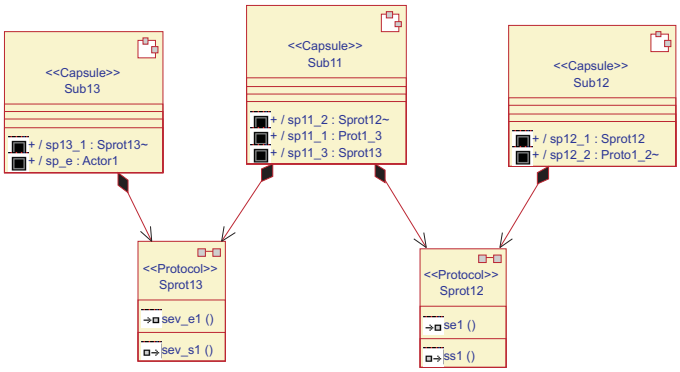


Figura 5.9: Diagrama de clase del *subsistema 1*

5.3.4.4. Modelado de la estructura interna

La estructura interna de cada subsistema es descrita por un diagrama de estructura compuesta, el cual muestra sus componentes y cómo están conectados. En la figura 5.3.4.4 se muestra la arquitectura interna del *subsistema 1*.

5.3.4.5. Comportamiento temporizado de los subsistemas

La última etapa del modelo consiste en definir el comportamiento temporizado de todos los componentes en el subsistema usando los diagramas de estados temporizadas. Cabe mencionar que los subsistemas primitivos son directamente representadas por diagramas de estados temporizados (para capturar el comportamiento del componente), dado que no hay estructura interna que describir.

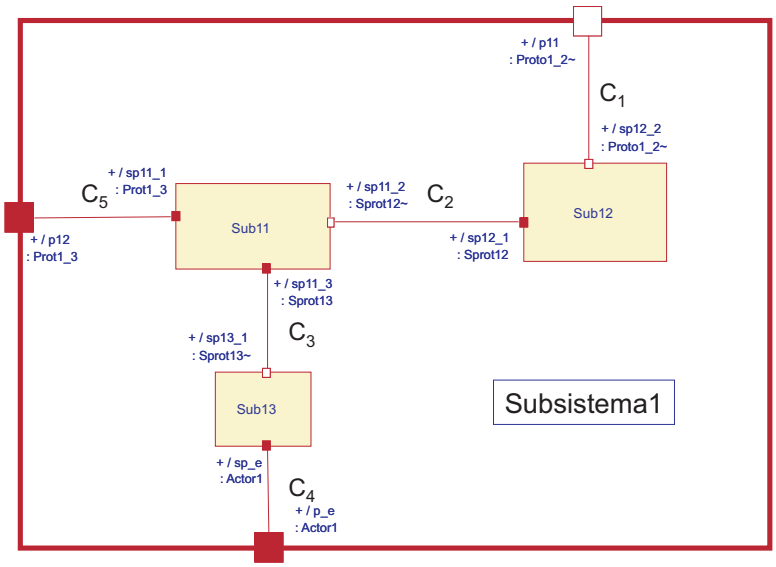


Figura 5.10: Estructura *interna del subsistema 1*

5.4. Especificación formal del sistema

5.4.1. Síntesis

La síntesis o composición del sistema representa la actividad de asociar y integrar los partes del sistema para que funcionen en conjunto. El comportamiento del sistema se deduce de la composición jerárquica del comportamiento de sus partes. La abstracción en este contexto se refiere al ocultamiento del comportamiento interno representando información irrelevante en el comportamiento de los partes o componentes.

La composición y la abstracción van de la mano en la especificación del comportamiento observable del sistema. Cada etapa del proceso de composición y abstracción tiende a formar componentes de caja negra que describen lo *que* debe hacer el sistema y alejándose de los detalles de la implementación, o sea, como el sistema debe comportarse en la realización de sus funciones, lo que da una vista más comprensible del sistema, sus componentes y sus responsabilidades.

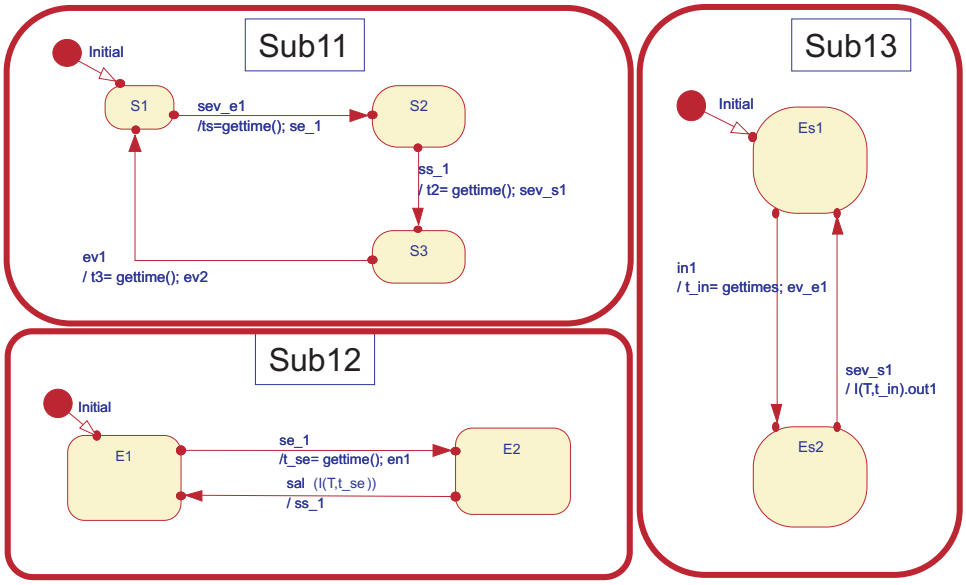


Figura 5.11: Diagramas de estado temporizado de los componentes del *subsistema 1*

5.4.2. El proceso bottom up de la especificación formal

Uno de los objetivos de MEDISTAM-RT es obtener el comportamiento detallado del sistema completo siguiendo una aproximación *composicional*. Esta consiste en la derivación del comportamiento de un sistema a partir del comportamiento de sus componentes. La figura 5.4.2 muestra esta aproximación.

Para ello, se ha definido un proceso de especificación que es en realidad un proceso de derivación de la especificación en términos de CSP+T a partir de los modelos UML-RT diseñados en la etapa de modelado del sistema. La derivación de esta especificación se lleva a cabo mediante la aplicación de las reglas de transformación expuestas en el capítulo 4.

El proceso de especificación se lleva a cabo de acuerdo a los siguientes pasos:

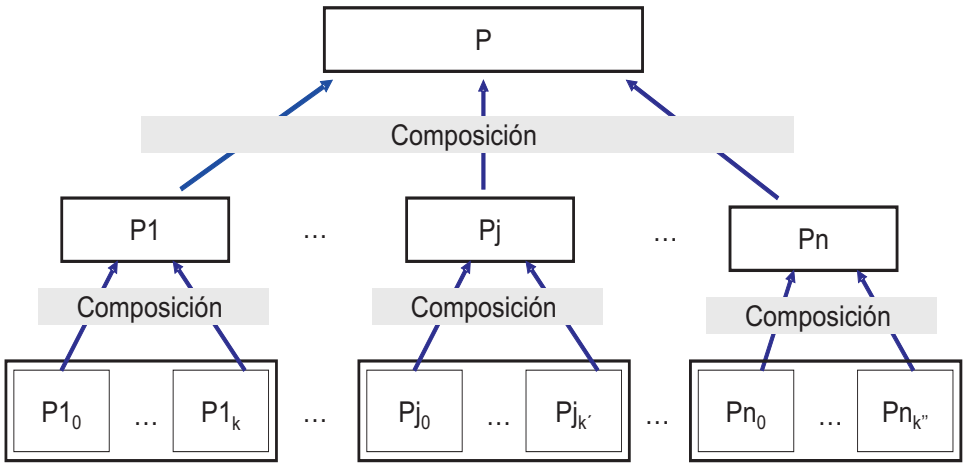


Figura 5.12: Estrategia composicional

Paso 1: Especificación formal del comportamiento temporizado de los subsistemas primitivos

Cada proceso representando un subsistema primitivo del sistema se especifica en términos de procesos CSP+T, en alto nivel de detalle, por el mapeo de entidades de los *diagramas de estados temporizados* en términos sintácticos de procesos CSP+T, mediante la aplicación del conjunto de reglas descritas en el capítulo 4, sección 4.2.1.1.

Este mapeo consiste en la especificación del comportamiento concreto de cada componente básico por medio de un proceso, que, a su vez, es descrito en términos de secuencia de eventos con restricciones de tiempo. Para realizar los pasos de especificación del modelo del ejemplo que se viene desarrollando desde la sección anterior, se comienza especificando los componentes básicos descritos por los diagramas de estados temporizados aplicando

La especificación formal CSP+T correspondiente al comportamiento de esos componentes se muestra en el cuadro 5.1.

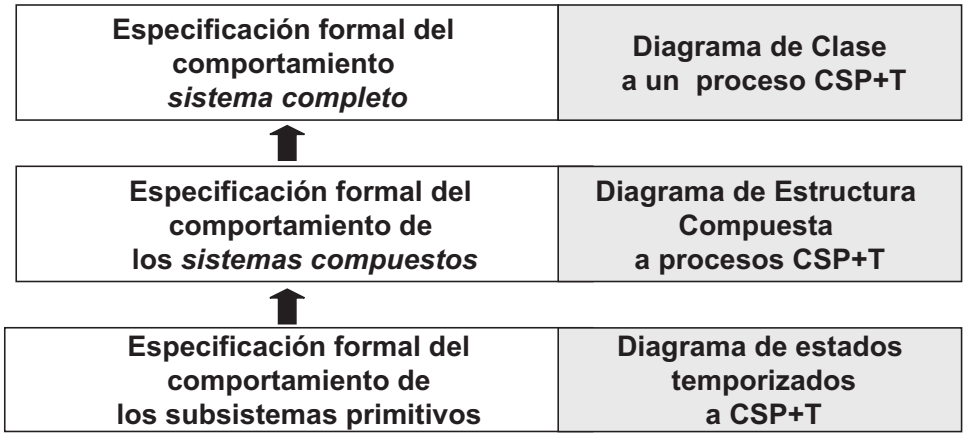


Figura 5.13: Estructura del proceso de especificación formal de sistemas

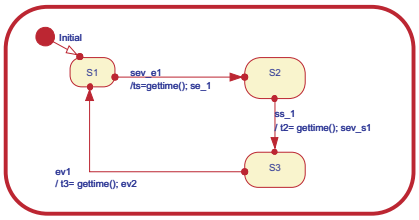
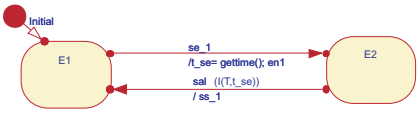
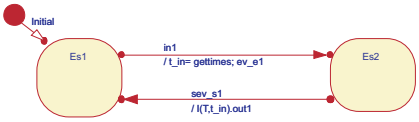
Paso 2: Descripción formal de los componentes compuestos del sistema

Con la finalidad de derivar la especificación formal de los subsistemas, se mapean las cápsulas compuestas, representadas mediante diagramas de estructura compuesta, a términos sintácticos de procesos CSP. Este paso permite la composición paralela de procesos descritos separadamente en el paso anterior. El procedimiento es ejecutado aplicando el conjunto de reglas de transformación presentadas en el capítulo 4.

Por ejemplo, la transformación del diagrama de estructura compuesta del *subsistema 1* en términos sintácticos CSP es realizada por la composición paralela de los procesos *Sub 11*, *sub12* y *Sub13*, descritos en el cuadro 5.1, aplicando los operadores de renombrado y ocultamiento, para obtener la especificación formal de la estructura de *Subsistema 1*, como se muestra en el cuadro 5.2.

El término de procesos que representa el sistema es entonces obtenido como una composición paralela de los tres componentes *sub11*, *Sub12*y *Sub13*. (**Sub11**$[R1]$ $\|$ **Sub12**$[R2]$ $\|$ **Sub13**$[R3]$).

La comunicación entre estos tres componentes se convierte en interna a

Cápsula	Diag. de estados temporizados	Especificación en CSP+T
Sub11		**Sub11** $= o.\star \rightarrow S_1$ $S1 = sev_1 \bowtie t_s \rightarrow se_1 \rightarrow S2$ $S2 = ss_1 \bowtie t2 \rightarrow sev_{s1} \rightarrow S3$ $S3 = ev1 \bowtie t3 \rightarrow ev2 \rightarrow S1$
Sub12		**Sub12** $= 0.\star \rightarrow E1$ $E1 = se_1 \bowtie t_s e \rightarrow en1 \rightarrow E2$ $E2 = I(T, t_{se}).sal \rightarrow ss_1 \rightarrow E1$
Sub13		**Sub13** $= 0.\star \rightarrow Es1$ $Es1 = in \bowtie t_i n \rightarrow ev_e 1 \rightarrow Es2$ $Es2 = sev_s 1 \rightarrow I(T, t_{in} f).out \rightarrow Es1$

Cuadro 5.1: Especificación formal del comportamiento de los componentes básicos de Sys

través del ocultamiento de todas las comunicaciones internas en el *Subsistema1*, estos son, los eventos que pasan a través de los conectores internos del mismo (C_3, y C_2).

C_2: representa la intersección de los alfabetos de comunicación de los dos procesos $Sub11$ y $Sub12$, este es el alfabeto de comunicación del protocolo $Sprot12$.

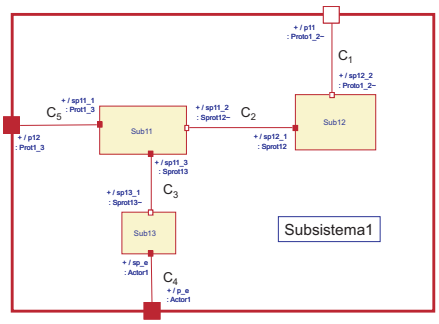
$$C_2 = \alpha Sprot12$$

$$= \{se1, ss1\}$$

De la misma manera, C_3 corresponde al alfabeto $\alpha(Sub11 \parallel Sub13)$, es decir, el alfabeto de comunicación del protocolo $Sprot13$.

$$C_3 = \alpha Sprot13$$

$$= \{sev_e 1, sev_s 1\}$$

Diagrama de Colaboración	Especificación en CSP+T
	$R_1 = \{sp11\_1 \mapsto C_5; sp11\_2 \mapsto C2;$ $sp11\_3 \mapsto C_3\}$ $R2 = \{sp12\_2 \mapsto C1;\ \ sp12\_1 \mapsto C_2\}$ $R_3 = \{sp11\_3 \mapsto C_3; sp\_e \mapsto C_4\}$ **Subsistema1** = (**Sub11**$[R1]$ $\|$ **Sub12**$[R2]$, $\|$ **Sub13**$[R3][C1 \mapsto p11]$, $[C5 \mapsto p12]) \setminus \{C2, C3\}$

Cuadro 5.2: Especificación formal del *subsistema*1 como una composición de subcápsulas

Paso 3: especificación formal del sistema completo.

Este paso consiste en la composición de los procesos descritos por el mapeo del diagrama inicial de contexto (diagrama de clase) en términos sintácticos de procesos CSP, mediante la aplicación del conjunto de reglas representadas en **el capítulo 4, sección 4.2.1.1**, con el fin de obtener la especificación completa del sistema.

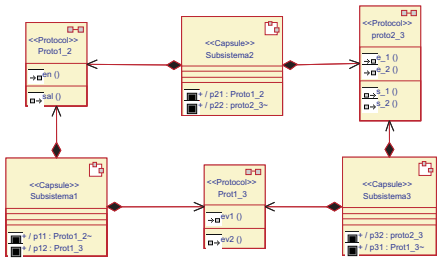
Para recopilar los procesos que representan todos los componentes del sistema, se transforma el diagrama de clases inicial (el diagrama de contexto), en términos de CSP.

El término de procesos que representa el sistema es entonces obtenido como una composición paralela de los tres componentes *Subsistema1*, *Subsistema2* y *Subsistema3*. El cuadro5.3, muestra la especificación formal del sistema completo.

5.5. Consistencia entre modelos y validación

5.5.1. ¿Que es la consistencia?

Los sistemas de tiempo real constan de múltiple componentes que se comunican a través de mensajes y que se ejecutan concurrentemente. El modelado de

Diagrama de Clases	Especificación en CSP+T
	$sistema$ =**Subsistema1**$[R1]$ ‖ **Subsistema2**$[R2]$ ‖ **Subsistema3**$[R3]$

Cuadro 5.3: Especificación formal del sistema completo Sys

diferentes perspectivas [5] y a diferentes niveles de abstracción es una técnica que ha mostrado ser eficiente para este tipo de sistema, por su capacidad de manejar el tamaño y la complejidad de los sistemas de tiempo real, dada la habilidad del estudio temprano de las propiedades del sistema a nivel de modelos, y el diseño de sistemas desde niveles de abstracción más altos a niveles más detallados. No obstante, el riesgo de inconsistencia entre los modelos del sistema generados es el precio a pagar para la utilización de este paradigma. La noción de inconsistencia se define en el diccionario Webster como:"una relación entre proposiciones que no puede ser verdadera al mismo tiempo", o también, "carencia de uniformidad harmoniosa entre las partes de la relación". Esta definición se puede aplicar en el contexto de UML, de ello la definición del concepto de consistencia se puede establecer como sigue: La propiedad de que diferentes submodelos no se contradigan mutuamente, es decir, los modelos se podrán considerar consistente si no existen contradicciones ni redundancias en las definiciones de los elementos y relaciones que las forman. Eso significa que existe una implementación que satisface los requerimientos expresados con los diferentes submodelos [GHKG02].

Se clasifica la consistencia de UML en diferentes tipos [ZLGS]:

- Intra-model(Consistencia horizontal): consistencia entre diferentes modelos al mismo nivel de abstracción.

- Inter-model (Consistencia vertical): consistencia entre modelos a diferentes niveles de abstracción.

[5] Estas perspectivas suelen ser entrelazadas

Otra clasificación divide la consistencia en [PBO07, MCB07]:

- Consistencia estática: se refiere a la consistencia en la sintaxis utilizado. Este tipo de consistencia se puede verificar con una inspección estática del modelo.
- Consistencia dinámica: concierne la consistencia en el comportamiento de los componentes de un sistema, y sólo se puede verificar después de la ejecución del sistema.

5.5.2. Diagrama de secuencia temporizada como herramienta para establecer la consistencia

Las vistas utilizadas en MEDISTAM-RT son vistas entrelazadas que tienen una serie de elementos relacionados. Los protocolos en el diagrama de clase encapsulan los eventos que pasan a través de los puertos y conectores definidos en el diagrama de estructura compuesta. Además, el comportamiento descrito con los diagramas de estados temporizados de los componentes deben satisfacer las restricciones impuestas por el comportamiento de los protocolos asignados a sus puertos. Estos son:

1. Las restricciones temporales.
2. El orden de paso de mensajes.

De otro lado, los diagramas de secuencia muestran las posibles interacciones entre componentes y proporciona una presentación estructurada del comportamiento de los componentes en forma de una secuencia de etapas secuenciales realizadas en el curso del tiempo. Aún más, esta secuencia representa una parte del comportamiento del sistema y una posible ejecución del diagrama de estado. Para ello, este componente brinda una vista general de las interacciones y paso de mensajes entre componentes.

Así pues, los diagramas de secuencia de los protocolos se pueden utilizar como una herramienta visual, para planificar tanto el paso de mensajes como sus restricciones temporales y establecer consistencia entre las diferentes etapas del proceso de modelado.

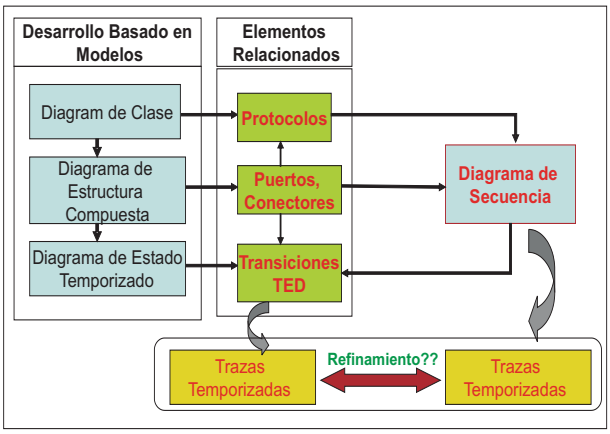


Figura 5.14: Diagrama de secuencia como elemento común

5.5.3. ¿Cómo establecer la consistencia temporal con diagramas de secuencia temporizados?

En este párrafo se va a presentar una serie de etapas a seguir para determinar y establecer las restricciones temporales que hay que marcar en un diagrama de secuencia [CBHM07], (véase la figura 5.15, como ejemplo).

1. El origen de tiempo en cada línea de vida esta marcado con el tiempo de ocurrencia del primer evento recibido en la línea de vida del diagrama de secuencia.

2. Durante una interacción entre dos componentes; el emisor transmite un mensaje y suspende sus acciones hasta que recibe un acuse de recepción del receptor del mensaje.

3. Cada mensaje m enviado entre dos componentes tiene una duración d, que representa el tiempo consumido entre el evento de envió ($!m$) y el evento de recepción ($?m$) del mensaje; y un tiempo de duración del mensaje de acuse de recepción (d_{ack}). En nuestro trabajo consideramos d_{ack} como un valor constante.

   a) De esa manera, el tiempo de la recepcion del evento (t_{re}) debe ser mayor que el tiempo del envío del mensaje t_s.

$$t_{re} = ts + d \ \& \ t_{re} > t_s$$

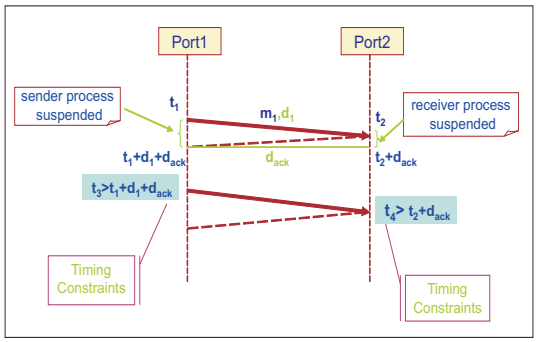


Figura 5.15: Diagrama de secuencia temporizado

b) Mientras que el intervalo $ts < t < ts{+}d{+}d_{ack} \equiv I(d{+}d_{ack}, ts)$ dura, el emisor se queda suspendido. Asimismo, dentro de este intervalo de tiempo, el emisor del mensaje no puede involucrarse en ninguna otra comunicación, a saber, emitir o recibir otros mensajes.

4. En el intervalo $I(d_{ack}, t_{re})$, el receptor del mensaje no puede aceptar establecer ninguna otra comunicación.

5.5.4. Reglas de consistencia

MEDISTAM-RT es una técnica de refinamiento paso a paso; en cada etapa de refinamiento la consistencia de una especificación junto con su abstracción precedente, debe ser formalmente verificada. Para ello, primero hay que definir un conjunto de propiedades que debe satisfacer la especificación para asegurar la consistencia vertical del modelo. Estas propiedades o condiciones dependen de los diagramas involucrados en las etapas del proceso del desarrollo:

1. La interconexión entre dos cápsulas en el diagrama de estructura compuesta debe ser conforme a la interconexión en el diagrama de clase (los puertos conservan sus roles).

2. Las comunicaciónes entre dos capsulas conectadas debe ser conforme a los protocolos.

3. El sistema debe ser libre de bloqueo; las restricciones temporales en todos los diagramas de estados deben ser consistentes.

5.5.5. Consistencia entre los diagramas de estados temporizados

El comportamiento de cada componente se describe separadamente con un diagrama de estado temporizado. Cuando estos componentes se componen en paralelo, sus diagramas de estados temporizados deben sincronizar. Un funcionamiento del sistema *libre de bloqueo* no se puede asegurar si no se establece *una consistencia temporal* en el comportamiento de los diferentes componentes comunicantes y una coordinación en su comportamiento. Esta tarea puede resultar muy tediosa si los diagramas de estados temporizados se elaboran razonando *separadamente* en cada componente.

El diagrama de secuencia sirve como un plano para proyectar el comportamiento de diferentes componentes. Cada *linea de vida* visualiza las interacciones del componente en una de sus interfaces o puertos.

Así pues, el diagrama de secuencia proporciona una vista que soporta el razonamiento sobre el comportamiento de los componentes en conjunto, y planificar el pasó de mensajes y sus restricciones temporales.

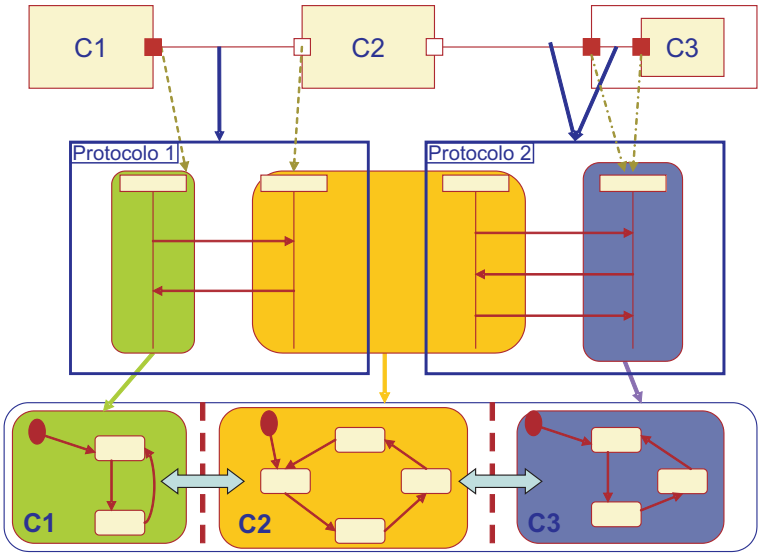


Figura 5.16: Diagrama de secuencia como elemento común

Con la transformación de los diagramas de secuencia temporizados a diagramas de estados temporizados, garantizamos la sincronización entre los

diferentes diagramas de estados. Para ello, se establece un conjunto de reglas de transformación de diagramas de secuencias a diagramas de estados temporizados parciales.

La demostración de estas reglas se lleva a cabo mediante la transformación de los diagramas de secuencias temporizados y los diagramas de estados temporizados a un dominio semántico común (Trazas temporizadas). Asimismo, para la transformación de un diagrama de secuencia (SD) a trazas temporizadas $Ttrazas(SD)$ se aplican la reglas de transformación presentadas en el capítulo 4, subsección 4.4.7. La transformación de un diagrama de estado temporizado a trazas se obtiene en dos etapas, primero se transforman los diagramas de estados D_C a procesos $CSP + T$ (con el mismo nombre (D_C)), aplicando las reglas presentadas en el capítulo 4, subsección 4.4.5 y luego se transforman los procesos obtenidos en la etapa anterior, aplicándoles la función semántica F_{TT} definida en el **capítulo 3**.

Anotaciones: Anotamos con:

1. $SD_{p,p'}$ un diagrama de secuencia temporizado que consta de dos *lineas de vida*, Una de un puerto X y otra de un puerto Y.

2. SD_p el diagrama de secuencia temporizado (con una sola linea de vida) de un componente C en el puerto p.

3. $D_{C,p}$ el diagrama de estado temporizado de un componente C en el puerto p.

Regla 1 sea m_0 el mensaje que inicializa el sistema, t_0 marca el origen de tiempo y T_0 marca la duración de initicialización. El tiempo de envío del primer mensaje m_1 por el elemento $p1$, debe ser mayor que $t_0 + T_0$, Esta restricción es importada al diagrama de estado del puerto $p1$ como el término $I(T, t_0 + T_0) \rightarrow !m_1$, que marca un retraso en el envío del mensaje m_1 a un intervalo de tiempo T, a partir del instante $t_0 + T_0$.

Dado que una interacción en el diagrama de secuencia representa una sincronización de comunicación entre los dos puertos $p1$ y $p2$, los dos puertos no podrán involucrarse en ninguna otra comunicación durante el *handshaking*. Por esta razón, el tiempo de envío o de recepción del siguiente mensaje m_2, debe ser superior al tiempo $t_1 + d + d_{ack}$, que es el tiempo de terminación de la cita (handshaking), d es la duración entre el envío y la recepción de los eventos

del mensaje m_2. Está restricción temporal se importa al diagrama de estado como $I(T, t_1 + d + d_{ack}) \rightarrow !m_2$ y $I(T, t_2 + d_{ack})$.

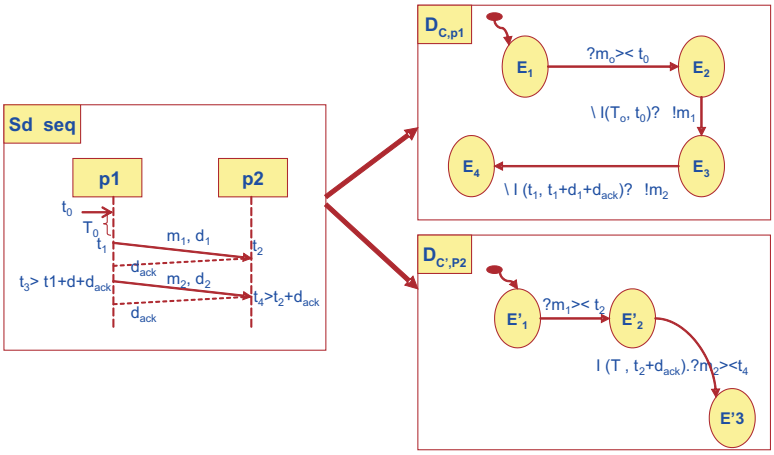


Figura 5.17: Regla 1

Con la transformación de los diagramas de secuencia $SD_{p1,p2}, S_{p1}$ y S_{p2} en la figura 5.17 a trazas temporizadas, obtenemos:

$$\begin{aligned} \mathbf{Ttraza(SD_{p1,p2})} &= (?m_0, t_0)^\frown (m_1, t_1, t_2)^\frown < m_2, t_3, t_4) \\ &= <?m_0, t_0)^\frown <!m_1, t_1)^\frown <?m_1, t_2)^\frown <!m_2, t_3) \\ &\quad ^\frown <?m2, t_4) \end{aligned} \tag{5.5.1}$$

Con :

$t_1 > t_0 + T_0$ *y* $t_3 > t_1 + d_1 + d_{ack}$

$t_4 b > t_2 + d_{ack}$

$$\mathbf{Ttraza(SD_{p1})} = (?m_0, t_0)^\frown(!m_1, t_1)^\frown(!m_2, t_3) \tag{5.5.2}$$

$$Con : t_1 > t_0 + T_0 \ y \ t_3 > t_1 + d_1 + d_{ack}$$

$$\mathbf{Ttraza(SD_{p2})} = (?m_1, t_2)^\frown(?m_2, t_4) \tag{5.5.3}$$

$$Con : t_4 > t_2 + d_{ack}$$

De la transformación de los diagramas de estados en la figura 5.17 a trazas temporizadas, obtenemos:

$$\mathbf{D_{C,p1}} = ?m_0 \bowtie t_0 \rightarrow (I(T_0, t_0) \rightarrow !m1) \rightarrow I(T, t_1 + d).!m_2 \rightarrow D_{C,p1}(E_4) \tag{5.5.4}$$

$$\mathbf{F_{TT}(D_{C,p1})} = (?m0, t0)^\frown(!m1, t1)^\frown(!m2, t3)^\frown F_{TT}(D_{C,p1}(E_4)) \tag{5.5.5}$$

$$Con : t_1 > t_0 + T_0 \ y \ t_3 > t_1 + d_1 + d_{ack}$$

$$\mathbf{D_{C',p2}} = ?m_1 \bowtie t_2 \rightarrow I(T_2, t_2 + d_{ack}).?m_2 \bowtie t_4 \rightarrow D_{C,p1}(E'_3) \tag{5.5.6}$$

$$\mathbf{F_{TT}(D_{C',p2})} = (?m_1, t_2)^\frown(?m_2, t_4)^\frown F_{TT}(D_{C,p2}(E_3)) \tag{5.5.7}$$

$$Con : t_4 > t_2 + d_{ack}$$

De las ecuaciones 5.5.2 y 5.5.5

$$F_{TT}(D_{C,p1}) \models_T Ttraza(SD_p1)$$

y de la ecuaciones 5.5.3 y 5.5.7 obtenemos

$$F_{TT}(D_{C',p2}) \models_T Ttraza(SD_p2)$$

De los términos de procesos $D_{C,p1}$ y $D_{C',p2}$, deducimos:

$$\mathbf{F_{TT}(D_{C,p1}||D_{C',p2}}) = (?m_0, t_0)^\frown(!m_1, t_1 > t_0 + T_0)^\frown(?m_1, t_2) \quad (5.5.8)$$

$$^\frown(!m_2, t_3 > t_0 + T_0 + d_1 + d_{ack})^\frown(?m_2, t_4 > t_2 + d_{ack})$$

$$^\frown F_{TT} D_{C,p1}(E4)||F_{TT} D_{C,p1}(E'3)$$

De las ecuaciones 5.5.1 y 5.5.8, deducimos:

$$F_{TT}(D_{C,p1}||D_{C',p2}) \models_T Ttraza(SD_{p1,p2}).$$

De ello, las comunicaciones del componente C y C' a través de los puertos $p1$ y $p2$ es conforme a la especificación del protocolo $Prot(P1)$.
Definición 1 Sea C un componente básico con 2 puertos $p1$ y $p2$.

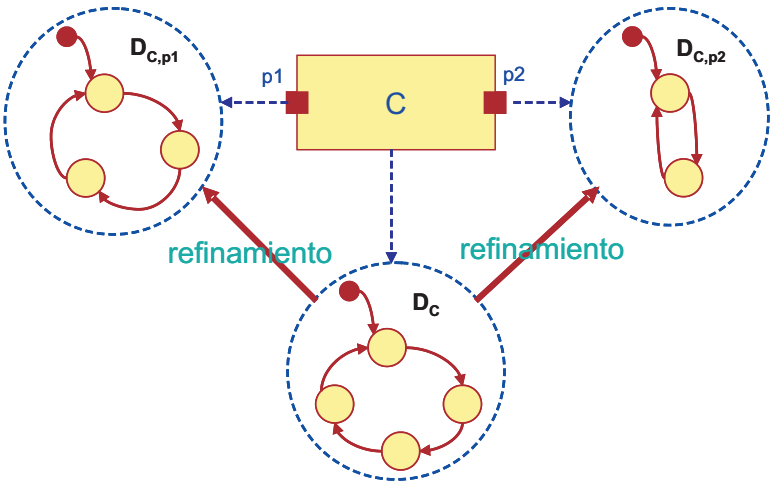


Figura 5.18: Un componente básico

Tenemos, $D_{C,p1} = D_C \setminus Ev_{e,\neg p1}$
$D_{C,p2} = D_C \setminus Ev2_{e,\neq p2}$
$\alpha(D_{C,p1}) = \alpha(D_C \setminus Ev_{e,\neg p1}) = Ev_{p1}$
$\alpha(D_{C,p2}) = \alpha(D_C \setminus Ev_{e,\neg p1}) = Ev_{p2}$
$\alpha(D_C) = ev_{p1} \cup ev_{p2} \cup In$. los Ev_{p1}, y Ev_{p2} representan los eventos del diagrama de estado temporizado de la Cápsula que pasan por el puerto $p1$ y $p2$,

respectivamente. Y In representa el conjunto de eventos internos de la cápsula C.

$$\begin{aligned}
F_{TT}(D_{C,p1}|||D_{C,p2}) =& \{tr \,|\, tr \restriction \alpha(D_{C,p1}) \in F_{TT}(D_C \setminus Ev_1) \\
& \wedge tr \restriction \alpha(D_{C,p1}) \in F_{TT}(D_C \setminus Ev_2)\} \\
=& \{tr \,|\, tr \restriction (Ev_{p1}) \in F_{TT}(D_C \setminus Ev_1) \\
& \wedge tr \restriction Ev_{p2} \in F_{TT}(D_C \setminus Ev_2) \\
=& \{tr \,|\, tr \restriction (Ev_{p1} \cup Ev_{p2}) \in F_{TT}(D_C \setminus Ev_1) \\
& \cup F_{TT}(D_C \setminus Ev_2)\} \\
=& \{tr \,|\, tr \restriction (Ev_{p1} \cup Ev_{p2}) \in F_{TT}(D_C \setminus In)\} \\
=& F_{TT}(D_C \setminus In)
\end{aligned}$$

De ello,

$$D_C \setminus In = D_{C,p1}|||D_{C,p2}$$

5.5.6. Consistencia en los diagramas de estructura compuesta

Un componente x_k con n puertos, se dice correcto si su especificación satisface las propiedades definidas en las n diagramas de secuencias temporizadas de los protocolos asociados a sus n puertos.

Como se ha demostrado antes, *los componentes básicos* diseñados con MEDISTAM-RT son componentes *correctos*, es decir, componentes que satisfacen las restricciones temporales y el orden de paso de mensajes requerido por los protocolos, lo que asegura la sincronización y la coordinacion entre los diferentes componentes básicos comunicantes. De ello, para cualquier componente básico X_k,

$$CSP + T(D_{Xk}) \models_T \bigwedge_{i:1\ldots n} SD_{p_i}$$

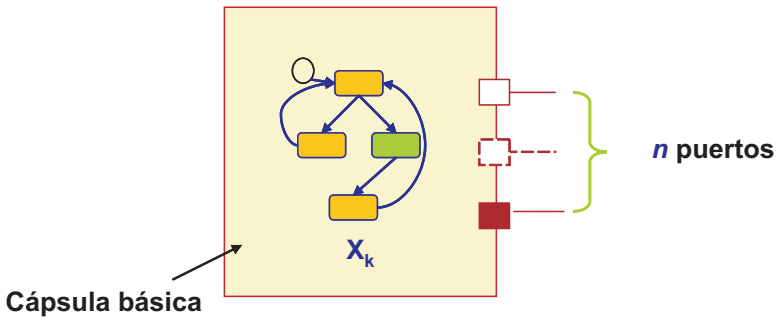


Figura 5.19: Comportamiento de un componente básico

Con la transformación del diagrama de estructura de un componente compuesto a términos de procesos CSP, obtenemos la especificación del comportamiento visible del mismo, en función de la especificación de sus subcomponentes.

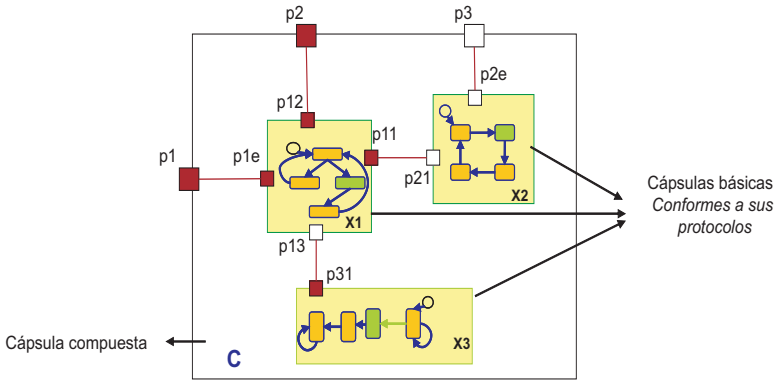


Figura 5.20: Un componente compuesto

$$D_C = (\|_{i:1...k} CSP + T(D_{Xk})) \setminus C_{int}$$

Con C_{int} es el conjunto de los conectores interiores en el componente X. D_C es correcta si $D_C \models_T \bigwedge_{i:1...3} SD_{p_3}$.

Capítulo 6

Caso de Estudio

6.1. Descripción de la celda de producción

Mediante la celda de producción propuesta queremos simular una parte de una planta industrial de fabricación que tiene como objetivo principal fabricar piezas metálicas con un tamaño, forma y acabado determinado como clavos, tornillos, placas, etc., para lo cual necesita deformar, perforar y/o cizallar el material sólido para que adopten la forma prevista. De los diferentes tipos de máquinas utilizadas habitualmente en la industria hemos considerado como máquina de modelado de nuestras piezas una prensadora especializada que como ventaja principal en la industria permite dar forma a las piezas sin eliminar material, o sea, sin producir virutas. En el modelo que hemos propuesto utilizaremos objetos cilíndricos que llegaran a una zona de la planta, un recipiente especifico, donde podrá actuar la prensadora dando forma por compresión.

El funcionamiento básico de la celda de producción pasa por las siguientes fases:

1. La cinta transportadora de entrada espera la llegada de un objeto para desplazarlo.

2. Cuando se coloca un objeto en la cinta transportadora de entrada, ésta

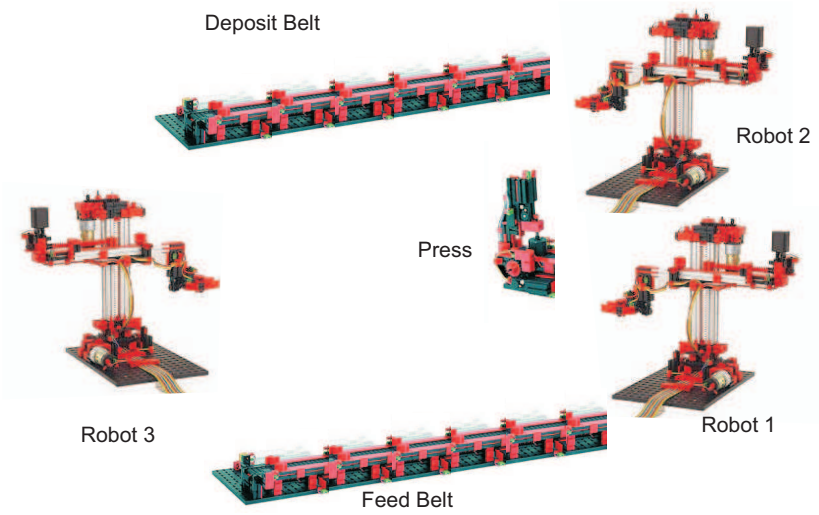


Figura 6.1: Celda de producción

se mueve y desplaza el objeto hasta el otro extremo del mismo. Posteriormente, avisa al sistema de que hay un objeto listo para ser manejado por el robot.

3. El robot que se encuentra inicialmente en una posición de espera, mueve todas las articulaciones simultáneamente (giro, movimiento vertical y horizontal, apertura de las pinzas) hasta situarse en la posición del objeto que se encuentra en la cinta transportadora de entrada. Recoge la pieza y la lleva al recipiente de la prensadora (que se encuentra a una altura distinta de las cintas transportadoras)

4. La prensa forja dicho objeto.

5. Posteriormente, un segundo robot recogerá la pieza ya modelada y la colocará en la cinta transportadora de salida (cinta transportadora 2 de la figura 6.1).

6. La cinta transportadora de salida debería desplazar las piezas modeladas a otra zona de la planta industrial para así efectuar algún otro tipo de manipulación o procesamiento con ellas.

7. Para poder realizar pruebas sobre la célula de producción sin la necesidad de la intervención del operador que tendría que recoger las piezas de una

cinta a la otra, hemos cerrado el sistema colocando un tercer robot al final de la cinta transportadora de salida, que se encargaría de mover la pieza de nuevo sobre la cinta transportadora de entrada.

6.2. Descripción de los elementos de la celda de producción

A continuación se describe el funcionamiento concreto de cada uno de los dispositivos del modelo físico del laboratorio señalando los tipos de movimientos válidos, el conjunto de sensores y actuadores de cada dispositivo, las restricciones que deben satisfacer, etc.

6.2.1. Los Robots

El robot industrial utilizado es un robot de juguete de tipo columna de la empresa Fischertechnik con cuatro grados de libertad: rotación en torno al eje (la columna) en sentido horario o antihorario (movimiento de giro), subir o bajar el brazo del robot sobre la columna para controlar su elevación (movimiento vertical), extender o retraer el brazo (movimiento horizontal), abrir y cerrar las pinzas del brazo para agarrar objetos.

Para que el robot pueda efectuar su trabajo requiere un conjunto de actuadores y sensores que nos permite accionar movimientos para cada articulación, controlar el desplazamiento de cada articulación y el establecimiento del sistema de referencia. Para efectuar la lectura de los sensores y envío de ordenes a los actuadores necesitamos comunicarnos al microprocesador "Intelligent Interface"mediante el puerto serie.

6.2.1.1. Actuadores

Para mover el robot cuenta con motores de corriente continúa en cada articulación que pueden realizar giros sobre un eje de rotación en cualquiera de sus dos sentidos. La construcción mecánica del robot permite utilizar los motores

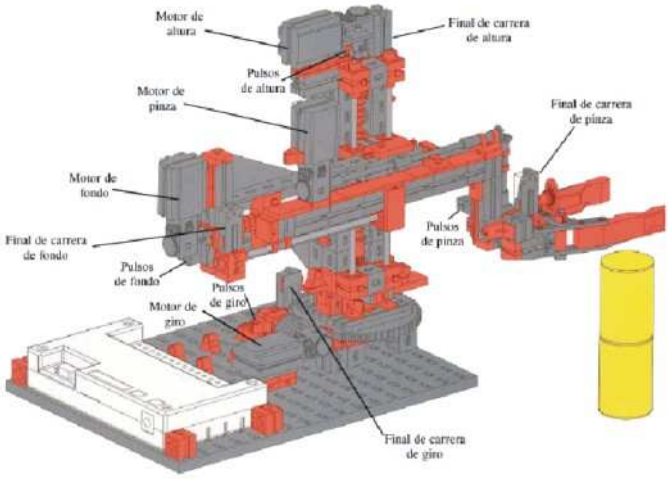


Figura 6.2: Diagrama de sensores y actuadores del robot de columna

para realizar movimientos de traslación y de rotación dependiendo de la articulación. En cualquier caso podemos elegir el sentido del movimiento tanto en el movimiento de traslación (izquierda o derecha) como en rotación (giro horario o antihorario).

6.2.1.2. Sensores

Para determinar la posición en la que se encuentra el robot en cada momento contamos con dos conjuntos de interruptores mecánicos pulsadores. Son sensores que proporcionan una señal positiva cada vez que se pulsa una muesca que se encuentra en uno de las partes del sensor. Cada grado de libertad del robot tiene dos interruptores pulsadores, uno para determinar una posición origen y el otro para determinar la posición actual del robot respecto al origen.

A continuación se explica cómo funciona cada uno de los sensores:

1. **Interruptor de fin de carrera:** Este interruptor pulsador se utiliza para determinar el origen de coordenadas se coloca al final de carrera en uno de los sentidos de movimiento de cada articulación; por defecto se encuentra siempre hacia el lado izquierdo si el movimiento es de traslación o en sentido antihorario si el movimiento es de giro.

2. **Interruptor contador de pulsaciones:** Este se utiliza para determinar la posición del robot respecto al origen de coordenadas para cada articulación.

6.2.2. Cinta transportadora

Las cintas transportadoras son elementos claves dentro de una cadena de producción dado que permiten transportar los objetos de un punto a otro de la planta industrial. En el modelo propuesto son necesarios dos cintas transportadoras como las de la figura 6.4. La cinta transportadora 1 es la encargada de proveer objetos al robot, mientras que la cinta transportadora 2 es donde el robot coloca los objetos que ha recogido de la cinta 1 y son transportadas a otro lugar de la planta.

Las cintas transportadoras se mueven mediante motores de corriente continua en el sentido de movimiento apropiado dependiendo de la cinta. La *cinta 1* se mueve a la izquierda mientras que la *cinta 2* se mueve a la derecha. Para cada cinta contamos con dos sensores *fototransistores*, uno para detectar la presencia del objeto al principio de la cinta y otro en la parte final. Cada sensor fototransistor se compone de dos partes, un transmisor de luz en un lado y un detector en el otro (figura 6.4). Cuando se coloca un objeto entre las dos partes, la luz no puede alcanzar el detector, y por tanto el sensor no proporciona señal alguna (respuesta negativa). En cambio si no colocamos ningún objeto entre las dos partes del *fototransistor*, éste produce una señal eléctrica (respuesta positiva).

Figura 6.3: Cinta transportadora

Cuando el robot coloca un objeto en el extremo de la cinta, el fototransistor

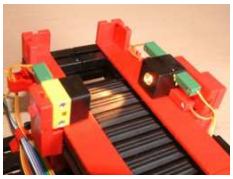


Figura 6.4: Fototransistor colocado en la cinta transportadora

informa que no hay paso de corriente, lo que indica la presencia del objeto. En ese instante de tiempo, el motor de la cinta transportadora lo pone en marcha en un sentido del movimiento dependiendo de la cinta transportadora. En el otro extremo de la cinta, otro fototransistor controla la llegada del objeto, para lo cual el sensor tiene que dar una respuesta negativa. Los movimientos que entran en juego en la cinta transportadora son:

- La cinta transportadora se inicia, comprueba los sensores y actuadores
- Después del proceso de inicio, el *fototransistor* del extremo 1 de la cinta transportadora se activa (o se atiende), mientras que el *fototransistor* del extremo 2 se desactiva (o no se atiende) y espera que alguien deje un objeto en el extremo 1.
- En cuanto el *fototransistor* del extremo 1 detecta la presencia de un objeto tiene que esperar un tiempo finito para que el robot pueda retirar el brazo del robot antes de proceder a su desplazamiento sobre la cinta.
- En ese momento se desactiva el *fototransistor* del extremo 1 y se activa el fototransistor del extremo 2 para proceder a continuación a accionar el motor de la cinta en el sentido del movimiento adecuado para que transporte la carga de un extremo al otro.
- La activación del *fototransistor* del extremo 2 en el paso anterior permite parar el motor cuando se haya detectado la llegada del objeto.
- A continuación espera a que otro robot recoja el objeto. Esto ocurrirá cuando el *fototransistor* del extremo 1 detecte la no presencia del objeto.
- En este momento se desactiva el *fototransistor* del extremo 2 y se vuelve a activar el *fototransistor* del extremo 1.

6.2.3. Prensa

La tarea de la prensa es forjar piezas de metal. La prensa consiste de dos plantas horizontales, donde la planta superior se mueve a lo largo del eje vertical. La prensa opera presionando la planta superior contra la planta inferior. La prensa tiene dos posiciones:

- La posición inferior, la prensa es descargada por Robot1.
- La posición Superior, la prensa se cierra forjando la pieza de metal.

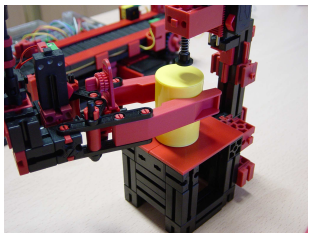


Figura 6.5: Prensa

Esta secuencia de procesamiento es ejecutada cíclicamente.

6.3. Modelado de la celda de producción:

6.3.1. Diagrama de contexto inicial de la celda de producción

El sistema funciona de manera automática, para ello, el sistema tiene un sólo actor, este es, el *operador* que se encarga de activar el sistema. La figura 6.6, muestra el diagrama de contexto inicial de la celda de producción.

6.3.2. Descomposición de la celda de producción:

La celda de producción se descompone en 6 subsistemas, tal y como se muestra en la figura 6.7:

1. *Cinta_Entrada*: que cuando está cargada manda una señal $fill$ al *Robot*1, para que la descargue.

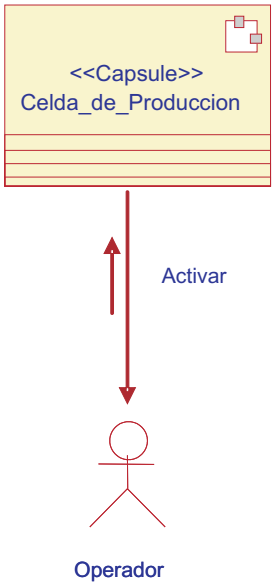


Figura 6.6: Diagrama de contexto inicial de la celda de producción

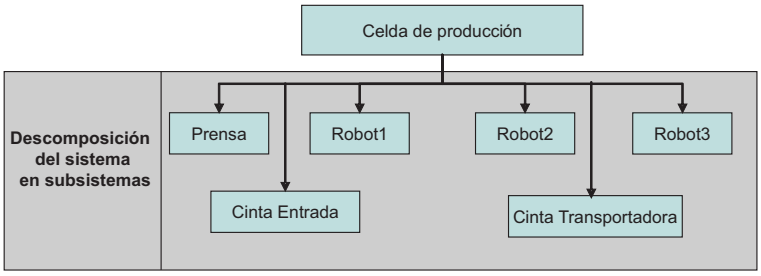


Figura 6.7: Descomposición de la celda de producción en subsistemas

2. *Robot*1: que se encarga de llevar los objetos de la *Cinta_Entrada* a la prensa.

3. *Prensa*: manda la señal *preparada* al *Robot*1 para que lo pueda descargar y forja los objetos cuando recibe la señal *forja* mandada por el *Robot1*.

4. *Robot2*: Cuando le llega la señal *ready* lleva los objetos forjados de *laprensa* y las deposita en *la Cinta de Transportadora*.

5. *Robot3*: mete los objetos a forjar en *la cinta_Entrada*.

6. *Cinta Transportadora*: transporta los objetos forjados.

6.4. Diagrama de contexto de la celda de producción

El *operador* del sistema interacciona con todos los subsistemas de la celda de producción, para activarlas. El diagrama de contexto en la figura 6.8 muestra el contexto y la arquitectura a alto nivel de la *celda de producción*, es decir, las interacciones con el exterior, y las cápsulas que componen la misma así que las señales que intercambian mutuamente para realizar la tarea de forjar objetos.

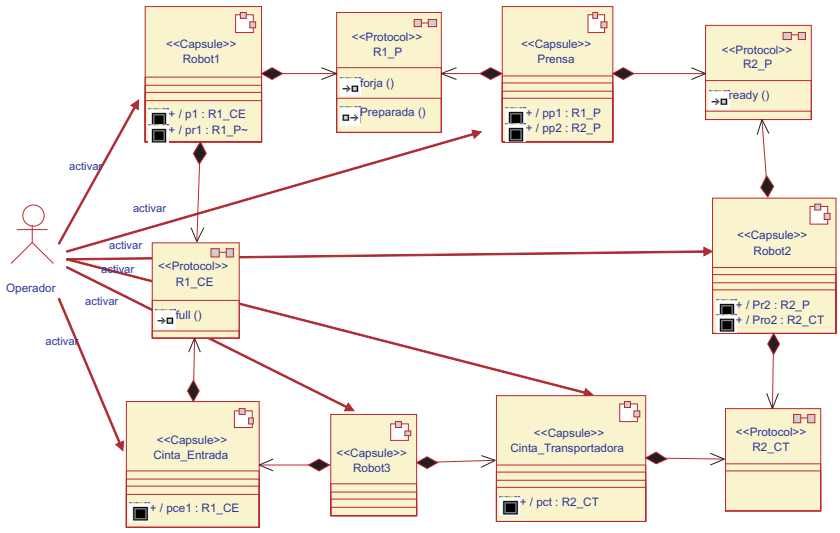


Figura 6.8: Diagrama de contexto de la celda de producción

6.5. Modelado del *Robot1*

Por motivos de claridad, se va a centrar en el modelado y la especificación de un elemento central de la celda de producción, éste es, el *Robot1*.

6.5.1. Descripción y requerimientos

El *Robot1* debe realizar dos funciones principales de la celda de producción:

1. Descargar la *Cinta-Entrada*.

2. Cargar la *prensa*.

Para realizar estas tareas, el *Robot1*, debe pasar por diferentes posiciones, la figura 6.10, muestra estas posiciones:

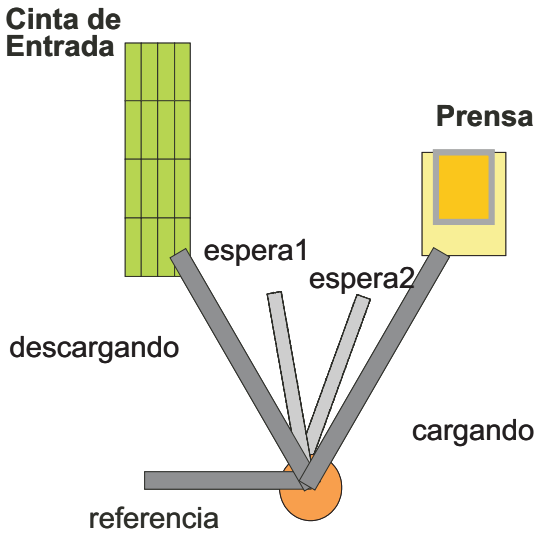


Figura 6.9: Esquema general con las posiciones generales de *Robot 1*

1. Al activar el *Robot1*, siempre se dirige a la posición de *Referencia*.

2. Luego, pasa directamente a una posición de espera (*Espera1*), donde está esperando que la cinta de entrada esté llena para descargarla.

3. Cuando le llega una señal de la cinta de entrada avisando de que esté llena, el robot se posiciona en la posición de *Descarga*

4. Cuando el robot coge el objeto de la cinta de Entrada, se posiciona en una posición de espera (*Espera2*), hasta que le avisa la prensa de que está preparada para cargarse.

5. Para cargar la prensa el robot se posiciona en la posición de *Carga*

Las **Coordenadas** de cada posición con respecto a todos los grados de libertad del robot se resumen en el cuadro 6.1

Posición	**Altura**	**Giro**	**Brazo**	**Pinza**
Referencia	h_pos1	g_pos1	retraído	abierta
Espera1	$h\_pos2 = h\_pos1$	g_pos2	retraído	abierta
Descarga	$h\_pos3 < h\_pos2$	g_pos3	extendido	cerrada
Espera2	$h\_posS > h\_pos3$	g_pos4	retraído	cerrada
Carga	$h\_pos5 = h\_posS$	g_pos5	extendido	abierta

Cuadro 6.1: Coordenadas de las posiciones de *Robot1*

6.5.1.1. Requerimientos de seguridad

1. El *Robot1* no puede soltar el objeto que carga, sólo si está en posiciones adecuadas (posición de *carga*).

2. Limitación en la movilidad de las máquinas;

   a) el brazo del *Robot1*, se puede romper si se extrae o se retrae más de lo posible.

   b) o bien, se puede romper si sube o baja más de lo posible.

   c) Además, el *Robot1*, no puede girar más de lo posible.

3. Riesgo de colisiones entre máquinas:

   a) el Robot sólo puede girar en la proximidad de la *prensa* si :

      1) la *prensa* está vacía,

      2) la *prensa* está en la posición superior.

      3) el *brazo* está retraído,

   b) el Robot sólo puede girar en la proximidad de la *Cinta_Entrada* si la posición de altura del mismo es superior a la altura de *la cinta_Entrada* $h\_pos3 + 1cm$.

6.5.2. Diagrama de contexto inicial del *Robot1*

Aplicando *MEDISTAM–RT*, primero se elabora el contexto de la cápsula *Robot1*, considerando todos los subsistemas de la *celda de producción*, que

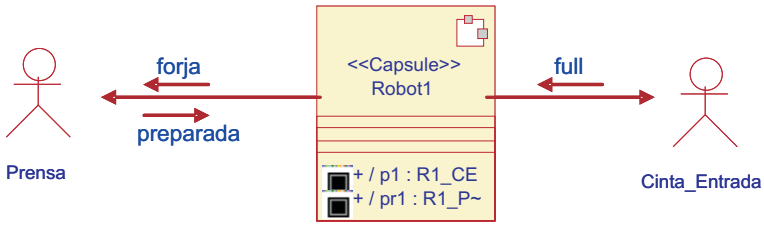


Figura 6.10: Diagrama de contexto de alto nivel de *Robot1*

interaccionan con el *Robot1*, como actores del mismo. La figura 6.10, muestra el diagrama de contexto de alto nivel del *Robot1*.

Del diagrama de contexto en la figura 6.8, se deduce que los actores del *Robot1*, son: la *prensa* y la *cinta_ entrada*. Las interacciones entre el *Robot1* y sus dos actores, se extraen de los protocolos $R1\_P$ y $R1\_CE$.

6.5.3. Descomposición del *Robot1*

Descomponemos el *Robot1* en dos cápsulas:

- Una cápsula compuesta, $Motor$: responsable de la coordinación del movimiento de todos los grados de libertad del *Robot*1, para que puedan sincronizar.

- Una cápsula primitiva, $Controlador\_R1$: interacciona con los actores del *Robot*1 y con el $Motor$.

6.5.4. Diagrama de contexto del *Robot1*

El controlador del robot $Controlador\_R1$, representa la interfaz del mismo con el exterior, es decir, las interacciones del *Robot1*, con sus actores, se establecen vía la interacciones de la prensa y la cinta de entrada con el $Controlador\_R1$. Este último interacciona también con el $Motor$ del *Robot1*. Las interacciones entre los mismos se encapsulan en el protocolo $RCRM$, tal y como muestra la figura 6.11.

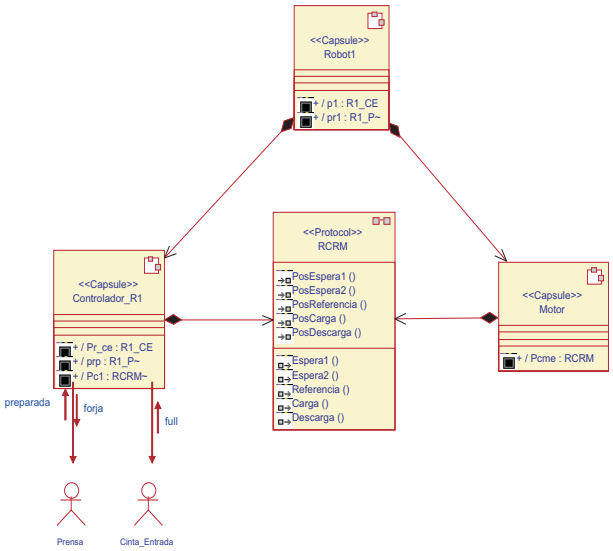


Figura 6.11: Diagrama de contexto de *Robot1*

6.5.5. Estructura del *Robot1*

La figura 6.15, muestra la estructura de *Robot1*, sus componentes y cómo están conectados.

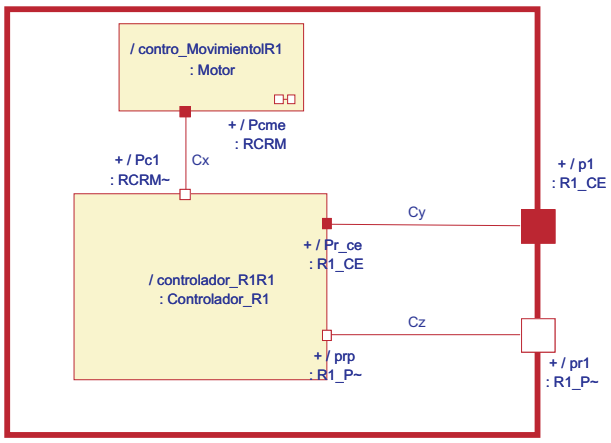


Figura 6.12: Diagrama de estructura compuesta de *Robot1*.

Como etapa de modelado siguiente, se diseña la cápsula compuesta de *Robot1*, esta es la cápsula motor.

6.5.6. Descomposición de *Motor*

La cápsula compuesta *Motor*, a su vez, se descompone en 7 cápsulas primitivas, tal y como se muestra en la figura 6.13:

1. *Rotador*: encargado del movimiento giratorio del *Robot1*.
2. *TensorExtensor*: encargado del movimiento horizontal del *Robot1*.
3. *Elevador*: encargado del movimiento vertical del brazo del *Robot1*.
4. *Pinza*: encargada de coger o soltar un objeto.
5. *SensorRotator*: informa de las posiciones del *Rotador*.
6. *SensorMov*: informa de las posiciones del *Elevador*.
7. *Controlador_Motor*: Coordina los movimientos de los diferentes grados de libertad del *Robot1*.

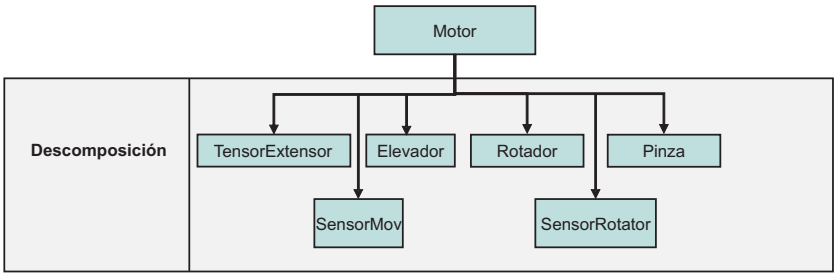


Figura 6.13: Descomposición de *Robot1*.

Requerimientos de seguridad de *Motor*

Para evitar colisiones, el comportamiento de la cápsula *Robot1*, debe satisfacer los siguientes requisitos:

1. Para recoger un objeto de la cinta de entrada: antes de moverse, el sistema debe estar en la posición de giro adecuada (g_pos3).
2. Para retirarse de la *cinta_entrada*, retraer el brazo antes de subir o girar.

3. Para soltar un objeto en la prensa:

   a) la prensa tiene que estar preparada (vacía y en la posición superior).

   b) segundo, sólo se puede abrir la pinza si el *motor de giro* y el *elevador* están en la posición de carga y el *brazo* extendido.

Especificación de los requerimientos de seguridad de *Motor*

1. El primer requerimiento se puede especificar en modelo de trazas temporizadas como sigue:

   $\forall tr \in F_{TT}(Motor)$ y $\forall s_1, s_2 \in \sigma(tr)$

   $$\mathbf{S_1(tr)} =, s_1 \frown moverDes \frown s_2 \frown g\_pos3 \textbf{ notin } \sigma(tr \uparrow [t(PosDescraga), t(cerrada)])$$

2. El segundo requerimiento se puede especificar en modelo de trazas temporizadas como sigue:

   $\forall tr \in F_{TT}(Motor)$,

   $$\mathbf{S_2(tr)} = \begin{cases} t(girarEsp2) > t(retrae) + 7 \\ \qquad\qquad y \\ t(girarEsp2) > t(subir) + 7 \end{cases}$$

3. No se puede soltar la pieza sólo si la prensa está preparada. Significa que después de cargar la pieza (cuando se cierra la pinza) la pinza no se puede *abrir* hasta que la prensa emita la señal preparada. Es decir el evento *abrir* no puede pertenecer a los elementos de la traza de *Motor* desde el momento de cerrar la pinza hasta el tiempo de ocurrencia de la señal *preparada*. Formalmente,

   $\forall tr \in F_{TT}(Motor)$

   $$\mathbf{S_3(tr)} = \quad abierta \textbf{ notin } \sigma(tr \uparrow [t(cerrada), t(preparada)])$$

Diagrama de contexto inicial de *Motor*

Al diseñar la cápsula compuesta $Motor$ de manera aislada de las otras cápsulas del sistema, se considera el $controlador\_R1$ como un actor del mismo. Las interacciones entre $Motor$ y el actor, se extraen del protocolo $RCRM$ en la figura 6.11.

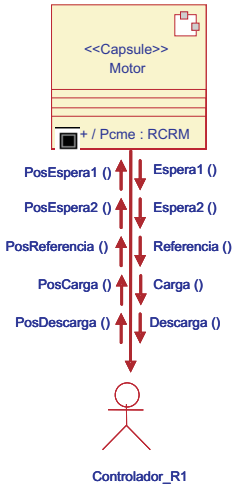


Figura 6.14: Diagrama de contexto inicial de *Motor*.

6.5.7. Arquitectura de *Motor*

La figura 6.15, muestra la arquitectura de la cápsula $Motor$ y las interacciones con el exterior. Las interacciones con el $Controlador\_R1$ se otorgan a la subcápsula $Controlador\_Motor$.

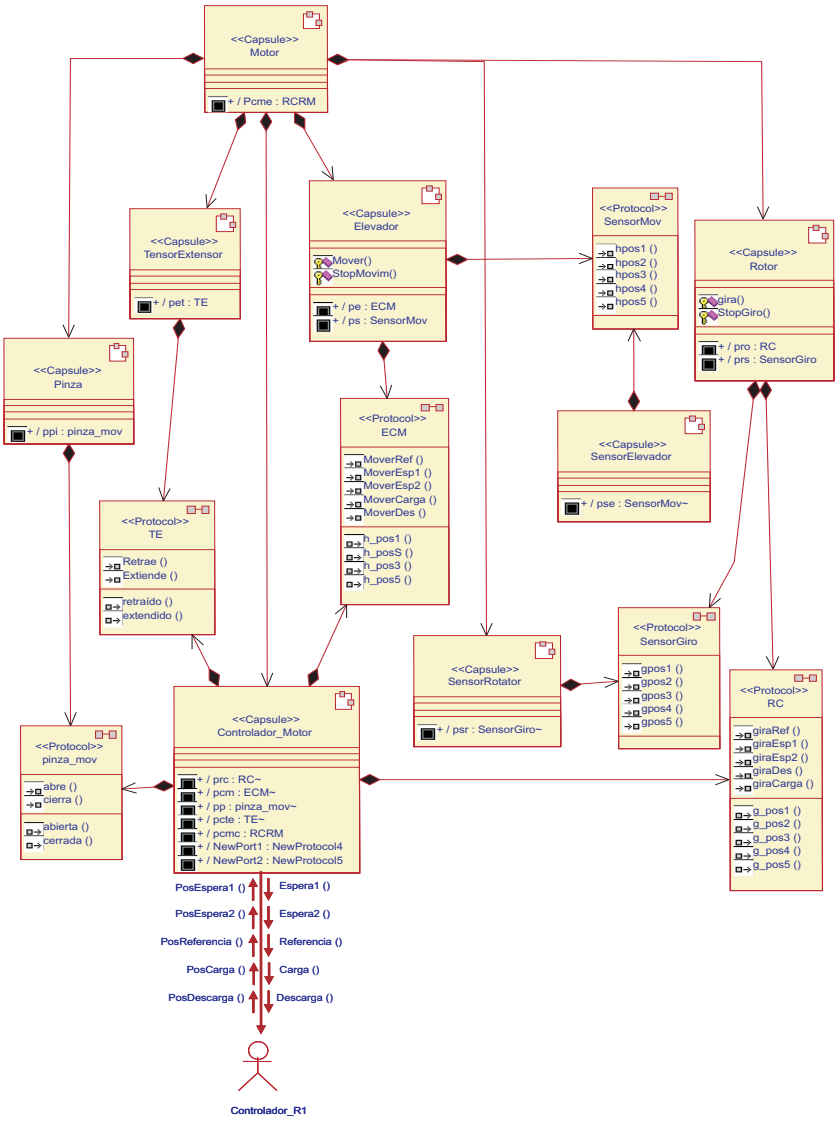


Figura 6.15: Diagrama de contexto de *Motor*.

6.5.8. Estructura interna de *Motor*

La figura 6.16, muestra el diagrama de estructura compuesta de la estructura interna de la cápsula *Motor*, que proporciona una vista de la estructura interna del mismo y los puertos de comunicaciones con el exterior (puerto P_{cme} con el role del protocolo básico *RCRM*)

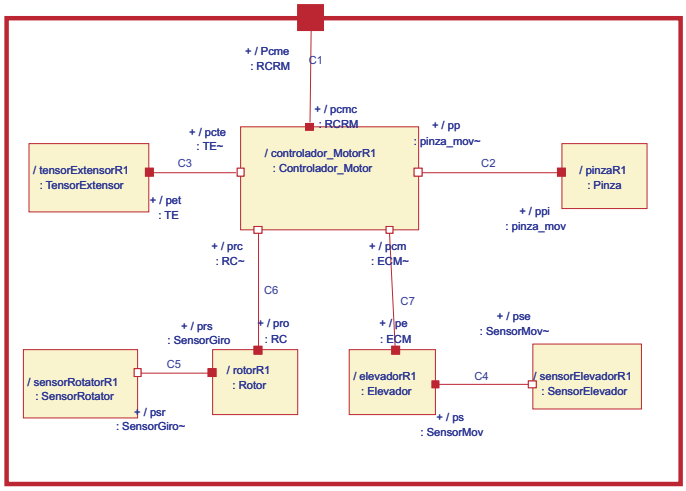


Figura 6.16: Diagrama de estructura compuesta de *Motor*.

6.5.9. Comportamiento de las cápsulas básicas del *Robot1*

La figura 6.17, muestra las interacciones de *Controlador_R1* con la *Prensa*, la *Cinta_Entrada* y con la cápsula *Motor*.

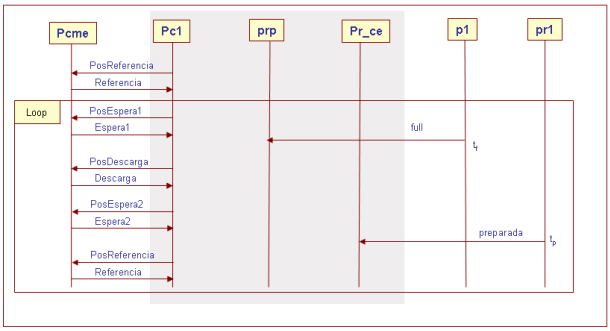


Figura 6.17: Diagrama de secuencia temporizado de *Controlador_R1*

A partir de la transformación del diagrama de secuencia temporizado en la figura 6.17, se diseña el diagrama de estado temporizado del *Controlador_R1* como se muestra en la figura 6.18.

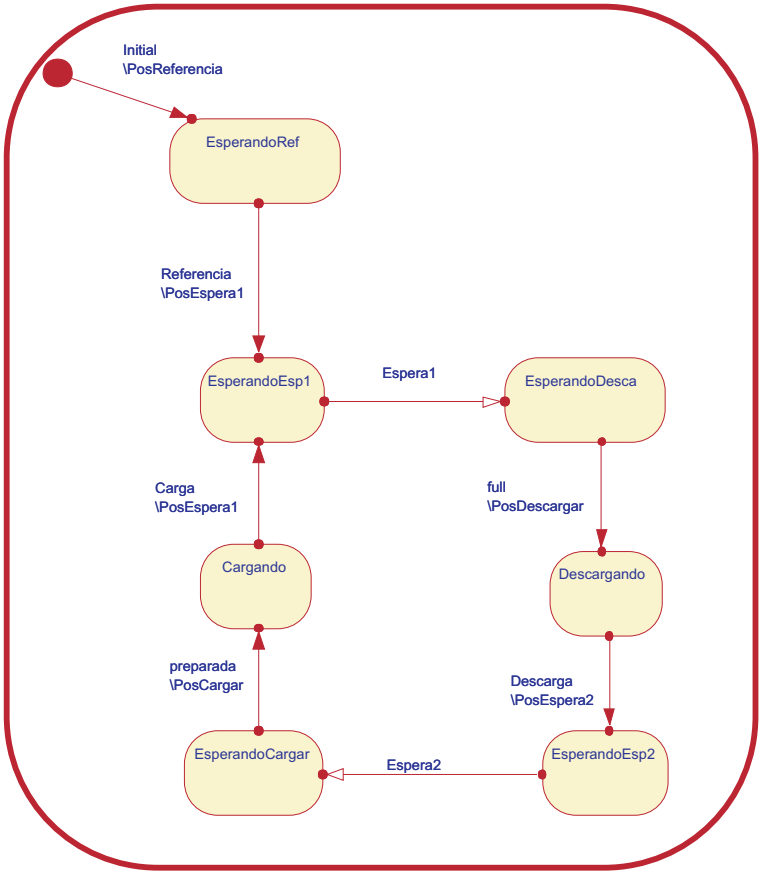


Figura 6.18: Diagrama de estados temporizados de *Controlador_R1*

Los diagramas de secuencia en las figuras 6.19 y 6.20, muestran las interacciones con el exterior y las interacciones entre los componentes de *Motor*.

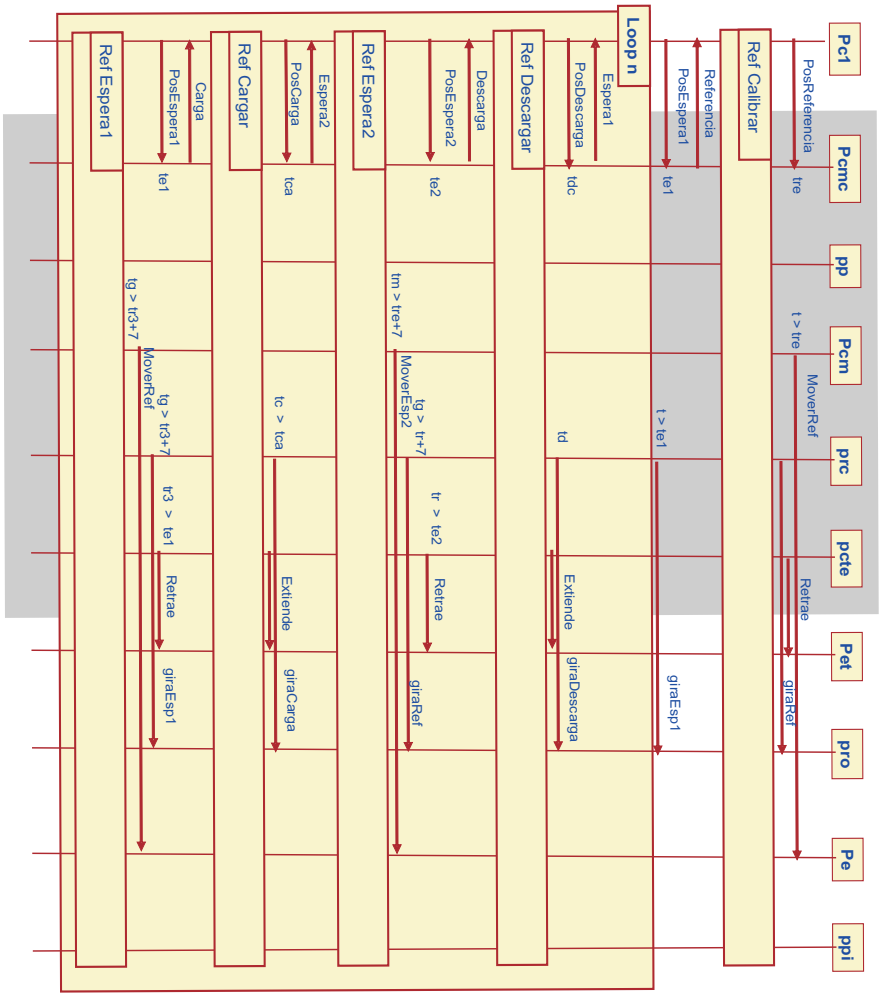


Figura 6.19: Diagrama de secuencia de *Motor*

A partir de la transformación del diagrama de secuencia temporizado en la figura 6.19, se diseña el diagrama de estado temporizado del *Controlador_Motor*, como se muestra en la figura 6.21.

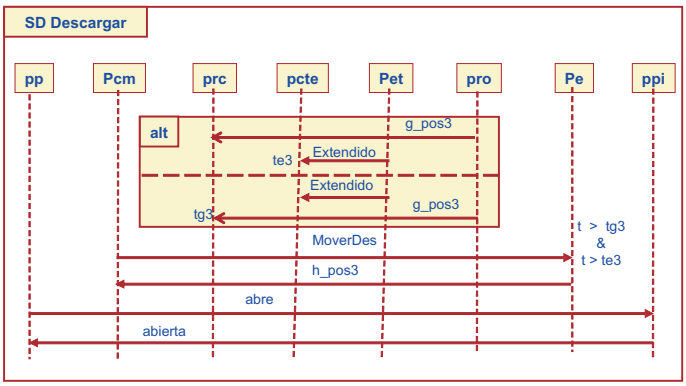


Figura 6.20: Escenario Descargar

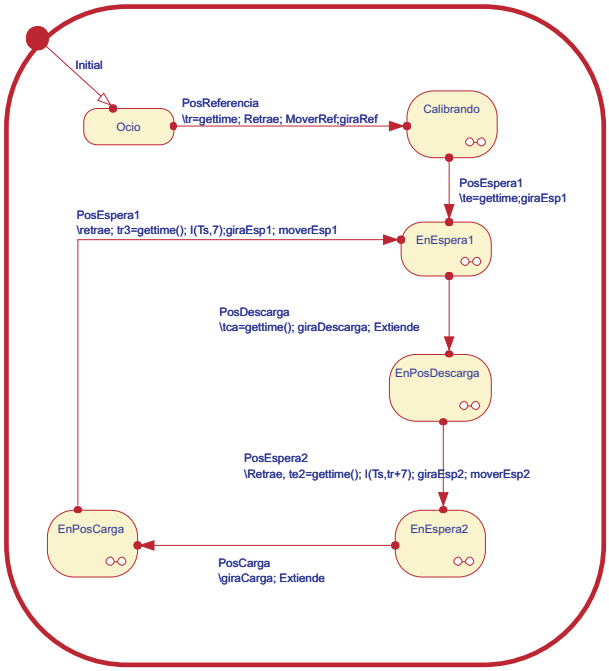


Figura 6.21: Diagrama de estado temporizado de *Controlador_Motor*

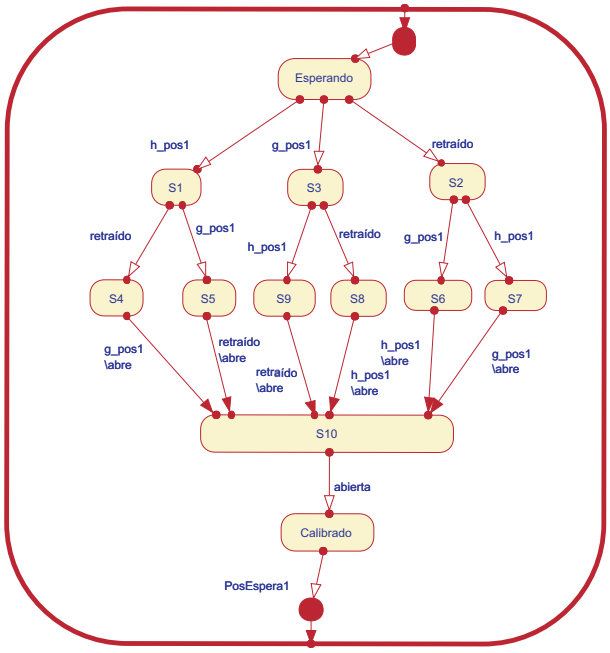


Figura 6.22: El estado compuesto *Calibrando*

El diagrama de estado temporizado de *Controlador_Motor* consta de cinco estados compuestos que se detallan a continuación en las figuras 6.22 hasta 6.25.

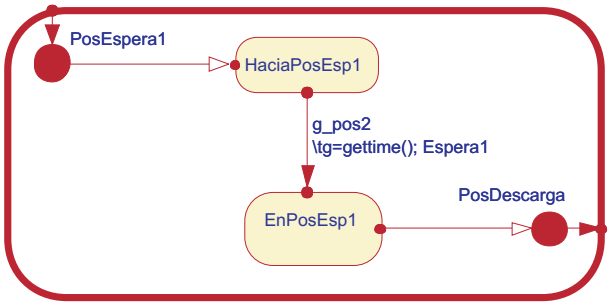


Figura 6.23: El estado compuesto *EnEspera*1

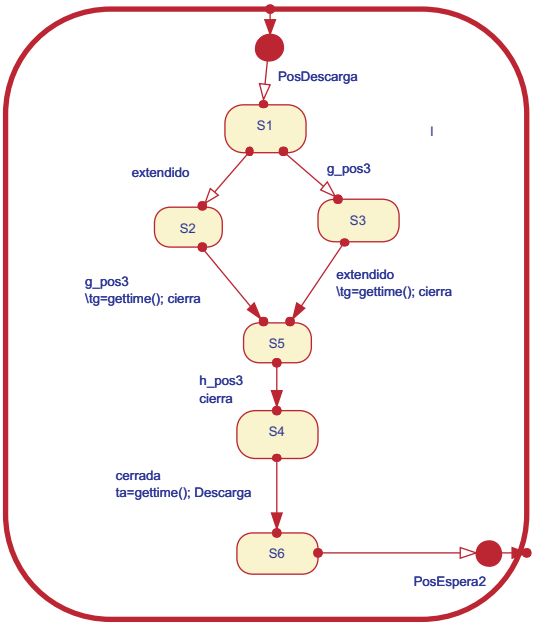


Figura 6.24: El estado compuesto *Descarga*

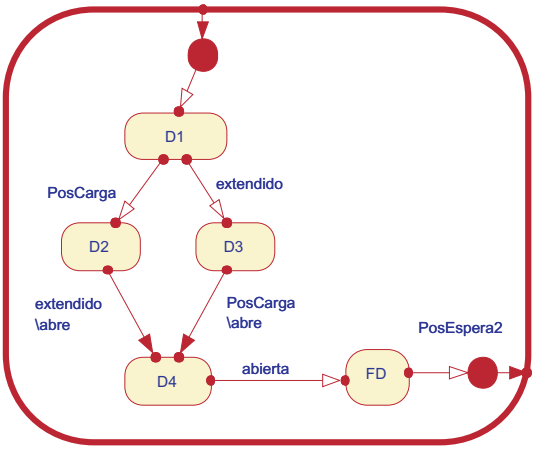


Figura 6.25: El estado compuesto *Carga*

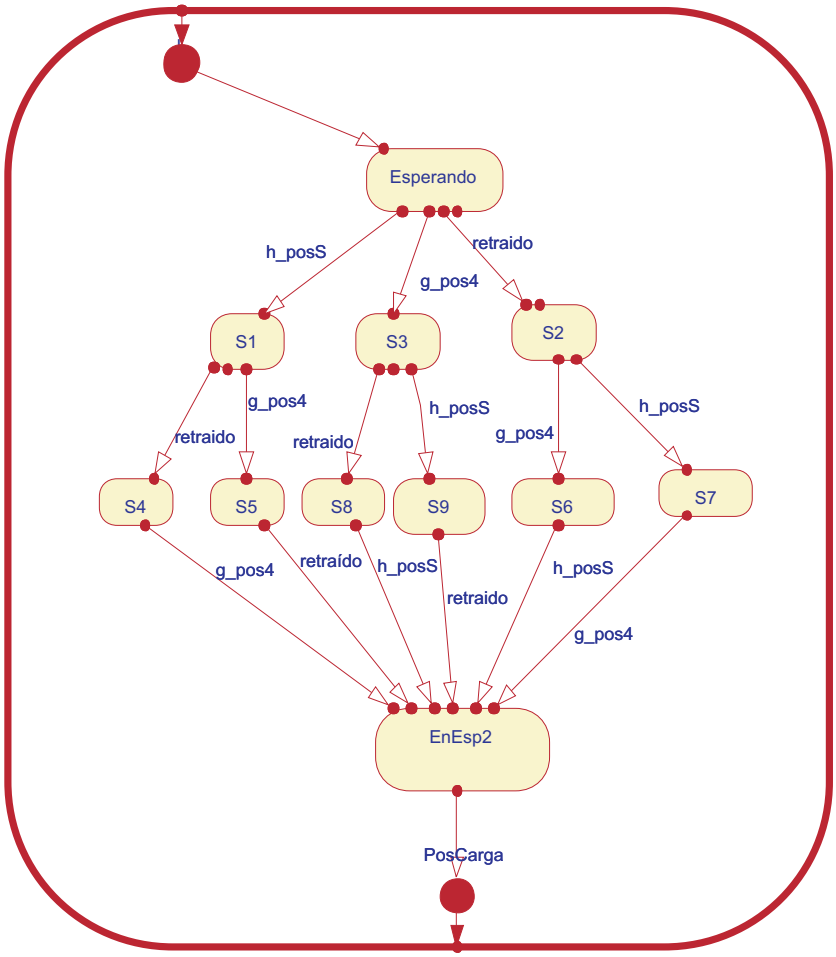


Figura 6.26: El estado compuesto *EnEspera*2

6.6. Especificación formal de la celda de producción

Para la derivación de la especificación formal del *Robot1* en términos sintácticos del CSP+T, a partir de los modelos UML-RT, se aplican las reglas de transformación establecidas en la sección 4.4, siguiendo el proceso de derivación de MEDISTAM–RT.

- **Etapa1:** La transformación de todos los diagramas de estados temporizados de las cápsulas básicas del *Robot1*, aplicando las reglas de transformación de los diagramas de estados temporizados a procesos CSP+T (establecidas en la sección 4.4.4).

  El resultado de esta derivación se muestra en los cuadros 6.2, 6.3 y 6.4. (Anotamos con $D_{CR}(I)$, el diagrama de estado temporizado de $Controlador\_R1$ a partir de su estado inicial I, y anotamos con $D_{CM}(I)$, el diagrama de estado temporizado de $Motor$ a partir de su estado inicial I))

- **Etapa2:** La especificación de la cápsula compuesta $Motor$ en función de sus componentes se obtiene mediante la transformación del diagrama de estructura compuesta de $Motor$ en la figura 6.16 a procesos CSP+T, aplicando las reglas de transformación establecidas en la sección 4.4.2. El resultado de esta transformación se muestra en la figura 6.5.

- **Etapa3:** La especificación formal del $Robot1$ en función de sus componentes, se obtiene mediante la transformación del diagrama de estructura compuesta del mismo a procesos CSP+T.

  El resultado de esta transformación se resume en el cuadro 6.6.

Comportamiento temporizado de $Controlador\_R1$

$$\mathbf{CSP + T(D_{CR}(I))}{=}initial.\star \rightarrow PosReferencia \rightarrow \mathbf{EsperandoRef}$$
$$EsperandoRef{=}Referencia \rightarrow PosEspera1 \rightarrow \mathbf{EsperandoEsp1}$$
$$EsperandoEsp1{=}Espera1 \rightarrow \mathbf{EsperandoDesca}$$
$$EsperandoDesca{=}full \rightarrow PosDescargar \rightarrow \mathbf{Descargando}$$
$$Descargando{=}Descarga \rightarrow PosEspera2 \rightarrow \mathbf{EsperandoEsp2}$$
$$EsperandoEsp2{=}Espera2 \rightarrow \mathbf{EsperandoCargar}$$
$$EsperandoCargar{=}preparada \rightarrow PosCargar \rightarrow \mathbf{Cargando}$$
$$Cargando{=}Carga \rightarrow PosEspera1 \rightarrow \mathbf{EsperandoEsp1}$$

La especificación del comportamiento de $Controlador\_Motor$ **en términos de procesos CSP+T**

$$\begin{aligned}
\mathbf{CSP + T(D_{CM}(I))}{=}&initial.\star \rightarrow PosReferencia \bowtie tr \rightarrow Retrae \\
&\rightarrow MoverRef \rightarrow giraRef \\
&\rightarrow \mathbf{D_{CM}}(\mathbf{inicial}(\mathbf{Calibrando})) \\
D_{CM}(final(Calibrando)){=}&PosEspera1 \bowtie te \rightarrow giraEsp1 \\
&\rightarrow \mathbf{D_{CM}}(\mathbf{inicial}(\mathbf{EnEspera1})) \\
D_{CM}(final(EnEspera1)){=}&PosDescarga \bowtie tca \rightarrow giraDescarga \\
&\rightarrow Extiende \\
&\rightarrow \mathbf{D_{CM}}(\mathbf{inicial}(\mathbf{EnPosDescarga})) \\
D_{CM}(final(EnPosDescarga)){=}&PosEspera2 \rightarrow Retrae \bowtie tr2 \rightarrow I(Ts, tr+7) \\
&\rightarrow giraEsp2 \rightarrow moverEsp2 \\
&\rightarrow \mathbf{D_{CM}}(\mathbf{inicial}(\mathbf{EnEspera2})) \\
D_{CM}(final(EnEspera2)){=}&PosCarga \rightarrow giraCarga \rightarrow Extiende \\
&\rightarrow \mathbf{D_{CM}}(\mathbf{inicial}(\mathbf{EnPosCarga}) \\
D_{CM}(final(EnPosCarga)){=}&PosEspera1 \rightarrow Retrae \bowtie tr3 \rightarrow I(Ts, tr) \\
&\rightarrow giraEsp1 \rightarrow moverEsp1 \\
&\rightarrow \mathbf{D_{CM}}(\mathbf{inicial}(\mathbf{EnEspera1}))
\end{aligned}$$

Cuadro 6.2: Transformación del diagrama de estado temporizado $Controlador\_R1$ y $Controlador\_Motor$ en términos de procesos CSP+T

El estado compuesto *Calibrando*

$$
\begin{aligned}
\mathbf{D_{CM}}(\mathbf{inicial}(\mathbf{Calibrando}) = & (h\_pos1 \rightarrow \mathbf{S1}) \\
& \Box \\
& (g\_pos1 \rightarrow \mathbf{S3}) \\
& \Box \\
& (retraido \rightarrow \ \mathbf{S2}) \\
S1 = & (retraido \rightarrow g\_pos1 \rightarrow \mathbf{S10}) \\
& \Box \\
& (g\_pos1 \rightarrow retraido \rightarrow \mathbf{S10}) \\
S2 = & (g\_pos1 \rightarrow h\_pos1 \rightarrow abre \rightarrow \mathbf{S10}) \\
& \Box \\
& (h\_pos1 \rightarrow g\_pos1 \rightarrow abre \rightarrow \mathbf{S10}) \\
S3 = & (retraido \rightarrow h\_pos1 \rightarrow \mathbf{S10}) \\
& \Box \\
& (h\_pos1 \rightarrow retraido \rightarrow \mathbf{S10}) \\
S10 = & abierta \rightarrow \mathbf{Calibrado} \\
\mathbf{Calibrado} = & \mathbf{D_{CM}}(\mathbf{final}(\mathbf{calibrando}))
\end{aligned}
$$

El estado compuesto *EnEspera1*

$$
\begin{aligned}
\mathbf{D_{CM}}(\mathbf{inicial}(\mathbf{EnEspera1})) = & g\_pos2 \bowtie tg \rightarrow Espera1 \\
& \rightarrow \mathbf{EnPosEsp1} \\
\mathbf{EnPosEsp1} = & \mathbf{D_{CM}}(\mathbf{final}(\mathbf{EnEspera1}))
\end{aligned}
$$

Cuadro 6.3: Transformación de los estados compuestos *Calibrando* y *En Espera1* en términos de procesos CSP+T

El estado compuesto *Descargando*

$$\begin{aligned}
\mathbf{D_{CM}}(\mathbf{inicial}(\mathbf{Descargando})) =& (g_pos3 \bowtie tg \rightarrow \mathbf{S3}) \\
& \Box \\
& (extendido \bowtie tex \rightarrow \mathbf{S2}) \\
S2 =& g_pos3 \bowtie tg \rightarrow cierra \rightarrow \mathbf{S5} \\
S3 =& extendido \bowtie tex \rightarrow cierra \rightarrow \mathbf{S5} \\
S5 =& cerrada \bowtie ta \rightarrow Descraga \rightarrow \mathbf{S6} \\
\mathbf{S6} =& \mathbf{D_{CM}}(\mathbf{final}(\mathbf{Descargando}))
\end{aligned}$$

El estado compuesto *EnEspera2*

$$\begin{aligned}
\mathbf{D_{CM}}((\mathbf{inicial}(\mathbf{EnEspera2})) =& (h_posS \rightarrow S1) \Box (g_pos4 \rightarrow S1) \\
& \Box retraido \rightarrow S2 \\
S1 =& (retraido \rightarrow g_pos4 \rightarrow \mathbf{EnEsp2}) \\
& \Box (g_pos4 \rightarrow retraido \rightarrow \mathbf{EnEsp2}) \\
S2 =& (h_posS \rightarrow g_pos4 \rightarrow \mathbf{EnEsp2}) \\
& \Box (g_pos4 \rightarrow h_posS \rightarrow \mathbf{EnEsp2}) \\
S3 =& (h_posS \rightarrow retraido \rightarrow \mathbf{EnEsp2}) \\
& \Box (h_posS \rightarrow retraido \rightarrow \mathbf{EnEsp2}) \\
\mathbf{EnEsp2} =& \mathbf{D_{CM}}(\mathbf{final}(\mathbf{EnEspera2}))
\end{aligned}$$

El estado compuesto *Cargando*

$$\begin{aligned}
\mathbf{D_{CM}}((\mathbf{inicial}(\mathbf{Cargando})) =& (extendido \rightarrow D3) \Box (PosCarga \rightarrow D2) \\
D2 =& PosCarga \rightarrow abre \rightarrow D4 \\
D3 =& extendido \rightarrow abre \rightarrow D4 \\
D4 =& abierta \rightarrow FD \\
\mathbf{FD} =& \mathbf{D_{CM}}(\mathbf{final}(\mathbf{cargando}))
\end{aligned}$$

Cuadro 6.4: Transformación de los estados compuestos *Descargando*, *EnEspera2* y *Cargando* a términos de procesos CSP+T

Transformación del diagrama de estructura compuesta de $Motor$ (figura 6.16) a CSP+T

$$R1 = \{pcmc \rightarrow C1; pcte \rightarrow C3; pp \rightarrow C2; pcm \rightarrow C7; Prc \rightarrow C6\}$$
$$R2 = \{pet \rightarrow C3\}, R3 = \{ppi \rightarrow C2\}, R4 = \{pe \rightarrow C7\}$$
$$R5 = \{pro \rightarrow C6; prs \rightarrow C5\}, R6 = \{psr \rightarrow C5\}, R7 = \{pse \rightarrow C4\}$$

$$\begin{aligned}\textbf{Motor} =&((Controlador\_Motor)[R1]|||(TensorExtensor)[R2]|||\\ &(Pinza)[R3]|||(Elevador)[R4]|||(Rotor)[R5]|||\\ &(SensorRotator)[R6]|||(SensorElevador)[R7]) \setminus CI\\ \textbf{CI} =&\{\alpha(TE) \cup \alpha(pinza\_mov) \cup \alpha(RC)\\ &\cup \alpha(ECM) \cup \alpha(SensorGiro)\}\end{aligned}$$

Cuadro 6.5: Especificación del comportamiento de $Motor$ en términos de procesos CSP+T

Transformación del diagrama de estructura compuesta de $Robot1$ (figura 6.15) a CSP+T

$$R8 = [Pcme \rightarrow Cx]$$
$$R9 = [Pc1 \rightarrow Cx; pr\_ce \rightarrow Cy; prp \rightarrow Cz]$$
$$R10 = [Cy \rightarrow p1; Cz \rightarrow pr1]$$

$$\begin{aligned}\textbf{Robot1} =&((Controlador\_R1)[R9]|||(Motor)[R8])[R10] \setminus C\\ \textbf{C} =&\alpha(RCRM)\end{aligned}$$

Cuadro 6.6: Especificación del comportamiento de $Robot1$ en términos de procesos CSP+T

Capítulo 7

Conclusiones y Trabajos Futuros

Con este trabajo de tesis se han obtenido las siguientes conclusiones:
La principal aportación de esta tesis se concreta en la definición de un nuevo enfoque metodológico, denominado MEDISTAM–RT, para el diseño y análisis correcto de sistemas de tiempo real, combinando métodos formales con métodos semiformales orientados a objetos basados en UML y UML-RT.

El marco metodológico MEDISTAM-RT permite realizar el modelado de sistemas de tiempo real basándose en modelos de UML y de UML-RT, cubriendo la perspectiva estática mediante los diagramas de clases y estructura-compuesta y la perspectiva dinámica mediante los diagramas de estados temporizados y diagramas de secuencias temporizados. Para dichas representaciones se han definido los correspondientes metamodelos, que permiten formalizar la semántica de los conceptos y elementos de modelado utilizados y sus relaciones.

La propuesta metodológica presentada en este trabajo de tesis permite el uso de UML para el modelado de los requisitos temporales y el establecimiento de la consistencia en el comportamiento de las diferentes cápsulas en las que aparezca recogida la especificación de un sistema de tiempo real. En particular,

se ha puesto especial énfasis en el establecimiento de la consistencia temporal. Además, la propuesta realizada posibilita la verificación formal del sistema mediante el lenguaje basado en álgebra de procesos CSP+T, partiendo del lenguaje semiformal UML-RT.

Para ello, por un lado, se ha enriquecido la sintaxis de UML-RT con anotaciones inspiradas en CSP+T para la descripción de las restricciones temporales. Asimismo, también se ha definido un conjunto de reglas de transformación que permiten derivar sistemáticamente una especificación formal en términos de procesos CSP+T, a partir de modelos de MEDISTAM–RT. Adicionalmente, se han definido técnicas para garantizar la consistencia entre los diferentes modelos del sistema objeto de diseño con la metodología.

Por otro lado, se ha definido una semántica denotacional para el lenguaje CSP con objeto de permitir especificar y verificar propiedades importantes de corrección de sistemas de tiempo real, como son las propiedades de seguridad y vivacidad.

La metodología propuesta define un método basado en transformación de modelos que, usando técnicas de refinamiento y abstracción, ayuda y guía a los diseñadores y analistas de un sistema en la obtención de una especificación formal que puede traducirse a CSP+T.

Existen varias líneas de trabajo que permiten seguir estudiando la especificación y verificación de los sistemas de tiempo real. Estas líneas son las siguientes:

- Definición de una semántica operacional para el lenguaje CSP+T en términos de autómatas temporizados que puedan ser aceptados por herramientas de evaluación de modelos (*Model checking*).

- Verificación *composicional*. Los mecanismos y estrategias utilizadas en MEDISTAM-RT, es decir, los mecanismos de abstracción y refinamiento junto con la descomposición del sistema en subsitemas, representan la base para la definición de una metodología de verificación *composicional*. Además, se puede definir un proceso formal de descomposición de requerimientos para permitir, por un lado, trazar los requerimientos entre los distintos modelos que contengan la especificación del sistema, y por otro lado, obtener al final de dicho proceso de descomposición los requerimientos básicos que deben satisfacer los componentes básicos

del sistema en construcción. De esta manera, la tarea de verificación de la corrección de un sistema complejo se puede reducir a la verificación de la corrección de sus componentes básicos y simples, lo que permite mitigar el problema de explosión de estados al aplicar evaluadores basados en *model checking*. Ello también permitiría la aplicación de métodos formales para la especificación y verificación de sistemas a gran escala.

- Automatización del proceso de desarrollo de sistemas aplicando MEDISTAM-RT. Para ello, hay que construir diferentes herramientas:

    1. Herramientas para la transformación automática de los modelos MEDISTAM-RT a especificaciones CSP+T.
    2. Herramientas para la generación automática de código a partir de especificaciones CSP+T.

Bibliografía

[Abr96] J.-R. Abrial, *The b-book: assigning programs to meanings*, Cambridge University Press, New York, USA, 1996.

[AD94] R. Alur and D. L. Dill, *A theory of timed automata*, Theor. Comput. Sci. **126** (1994), no. 2, 183–235.

[AFH91] R. Alur, T. Feder, and T. A. Henzinger, *The benefits of relaxing punctuality*, Tech. report, Stanford, CA, USA, 1991.

[AG97] R. Allen and D. Garlan, *A formal basis for architectural connection*, ACM Trans. Softw. Eng. Methodol. **6** (1997), no. 3, 213–249.

[AH93] R. Alur and T.A. Henzinger, *Real-time logics: complexity and expressiveness*, Inf. Comput. **104** (1993), no. 1, 35–77.

[AKZ96] M. Awad, J. Kuusela, and Jürgen Ziegler, *Object-oriented technology for real-time systems: a practical approach using omt and fusion*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[AM92] A.Combes and J. Mcdermid, *Specifying temporal requirements for distributed real-time systems in z*, Software Engineering Journal, 1992, pp. 273–283.

[Bae05] J. C. M. Baeten, *A brief history of process algebra*, Theor. Comput. Sci. **335** (2005), no. 2-3, 131–146.

[BB87] T. Bolognesi and E. Brinksma, *Introduction to the iso specification language lotos*, Comput. Netw. ISDN Syst. **14** (1987), no. 1, 25–59.

[BCH06] K. Benghazi, M.I. Capel, and J.A. Holgado, *Design of real–time systems by systematic transformation of UML–RT models into simple timed process algebra system specifications*, Proc. 8th International Conference on Enterprise Information Systems (ICEIS 2006) (Setúbal, Portugal), INSTICC Press, 2006, pp. 290–297.

[BCHM07] K. Benghazi, M.I. Capel, J.A. Holgado, and L. E. Mendoza, *A methodological approach to the formal specification of real–time systems by transformation of UML–RT design models*, Science of Computer Programming **65** (2007), no. 1, 41–56.

[BDKTV99] F. M. T. Brazier, B. Dunin-Keplicz, J. Treur, and R. Verbrugge, *Modelling internal dynamic behaviour of bdi agents*, Selected papers from the ESPRIT Project ModelAge Final Workshop on Formal Models of Agents (London, UK), Springer-Verlag, 1999, pp. 36–56.

[BF98] JM. Bruel and R. B. France, *Transforming uml models to formal specifications*, OOPSLA'98, Springer, 1998.

[BJ07] P. Baker and C. Jervis, *Early uml model testing using ttcn-3 and the uml testing profile*, TAICPART-MUTATION '07: Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION (Washington, DC, USA), IEEE Computer Society, 2007, pp. 47–54.

[BP02] P. J. Brooke and R. F. Paige, *The design of a tool-supported graphical notation for timed csp*, IFM '02: Proceedings of the Third International Conference on Integrated Formal Methods (London, UK), Springer-Verlag, 2002, pp. 299–318.

[BRJ99] G. Booch, J. Rumbaugh, and I. Jacobson, *The unified modeling language user guide*, Addison-Wesley, Massachusetts, USA, 1999.

[But97] G.C. Buttazzo, *Hard real-time computing systems: Predictable scheduling algorithms and applications*, Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[BW01] A. Burns and A. J. Wellings, *Real-time systems and programming languages: Ada 95, real-time java, and real-time posix*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[CBHM07] M.I. Capel, K. Benghazi, J.A. Holgado, and L.E. Mendoza, *An interpretation of behavioural consistency of UML–RT diagram in terms of CSP+T*, Proc. 5th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems (MSVVEIS 2007) (Setúbal, Portugal), INSTICC Press, 2007, pp. 74–83.

[CDKM02] F. Cottet, J. Delacroix, C. Kaiser, and Zoubir Mammeri, *Scheduling in real-time systems*, John Wiley and Sons, 2002.

[CMC06] L. Chung, W. Ma, and K. Cooper, *Requirements elicitation through model-driven evaluation of software components*, ICCBSS '06Proceedings of the Fifth International Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems, 2006, p. 187.

[CSLO00] J.-P. Courtiat, C. A. Santos, C. Lohr, and B. Outtaj, *Experience with rt-lotos, a temporal extension of the lotos formal description technique*, Computer Communications **23** (2000), no. 12.

[CW96] E. M. Clarke and J. M. Wing, *Formal methods: state of the art and future directions*, ACM Comput. Surv. **28** (1996), no. 4, 626–643.

[DL91] J. Dick and J. Loubersac, *Integrating structured and formal methods: A visual approach to vdm*, ESEC '91: Proceedings of the 3rd European Software Engineering Conference (London, UK), Springer-Verlag, 1991, pp. 37–59.

[Dou97] B. P. Douglass, *Real-time uml: Developing efficient objects for embedded systems*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[Dou99] ———, *Doing hard time: developing real-time systems with uml, objects, frameworks, and patterns*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[DPSS80] D.Gabbay, A. Pnueli, S. Shelah, and J. Stavi, *On the temporal analysis of fairness*, POPL '80: Proceedings of the 7th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (New York, NY, USA), ACM, 1980, pp. 163–173.

[EBHC06] A. Escámez, K. Benghazi, J. A. Holgado, and M. I. Capel, *Cspjade: Architectural driven development of complex embedded system software using a csp paradigm based generation tool code*, MSVVEIS, 2006, pp. 128–133.

[ELA90] E.Harel, O. Lichtenstein, and A.Pnueli, *Explicit clock temporal logic*, Procceding of 5th Annual Symposmm on Logic in Computer Science (New York, USA), IEEE Computer Society Press,, 1990, pp. 402–413.

[EMSS91] E. A. Emerson, A. K. Mok, A. P. Sistla, and J. Srinivasan, *Quantitative temporal reasoning*, CAV '90: Proceedings of the 2nd International Workshop on Computer Aided Verification (London, UK), Springer-Verlag, 1991, pp. 136–145.

[FA00] J.L. Fernández and A.T. Álvarez, *Can intuition become rigorous? foundations for uml model verification tools*, ISSRE '00: Proceedings of the 11th International Symposium on Software Reliability Engineering (ISSRE'00) (Washington, DC, USA), IEEE Computer Society, 2000, p. 344.

[FA01] J.L. Fernández and A. T. Álvarez, *Seamless formalizing the uml semantics through metamodels*, 224–248.

[Fey97] K. Feyerabend, *A visual formalism for real time requirement specifications*, In Transformation-Based Reactive System Development, number 1231 in LNCS, Springer Verlag, 1997, pp. 156–168.

[FKV94] M. D. Fraser, K. Kumar, and V. K. Vaishnavi, *Strategies for incorporating formal specifications in software development*, Commun. ACM **37** (1994), no. 10, 74–86.

[FOW01] C. Fischer, E-R. Olderog, and H. Wehrheim, *A CSP view on UML-RT structure diagram, lecture notes in computer science 2029: Fundamental approaches to software engineering*, pp. 91–108, Springer-Verlag, Berlin, 2001.

[GHKG02] G.Engels, R. Heckel, J. M. Küster, and L. Groenewegen, *Consistency-preserving model evolution through transformations*, UML '02: Proceedings of the 5th International Conference on The Unified Modeling Language (London, UK), Springer-Verlag, 2002, pp. 212–226.

[Gom93] H. Gomaa, *Software design methods for concurrent and real-time systems*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1993.

[Gom00] ———, *Designing concurrent, distributed, and real-time applications with uml*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[Hal90] A. Hall, *Seven myths of formal methods*, IEEE Softw. **7** (1990), no. 5, 11–19.

[Har87] D. Harel, *Statecharts: A visual formalism for complex systems*, Sci. Comput. Program. **8** (1987), no. 3, 231–274.

[Her99] D. Herzberg, *UML-RT as a candidate for modeling embedded real-time systems in the telecommunication domain, lecture notes in computer science 1723: Uml '99*, pp. 330–338, Springer-Verlag, Berlin, 1999.

[H.K91] H.Kopetz, *Event-triggered versus time-triggered real-time systems*, Proceedings of the International Workshop on Operating Systems of the 90s and Beyond (London, UK), Springer-Verlag, 1991, pp. 87–101.

[HKU02] R. M. Hierons, T.H. Kim, and H. Ural, *Expanding an extended finite state machine to aid testability*, COMPSAC '02: Proceedings of the 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment (Washington, USA), IEEE Computer Society, 2002, pp. 334–342.

[Hoa78] C. A. R. Hoare, *Some properties of predicate transformers*, J. ACM **25** (1978), no. 3, 461–480.

[Hoa85] C.A.R. Hoare, *Communicating sequential processes*, International Series in Computer Science, Prentice–Hall International Ltd., Hertfordshire UK, 1985.

[HP87] D. J. Hatley and I.A. Pirbhai, *Strategies for real-time system specification*, Dorset House Publishing Co., Inc., New York, NY, USA, 1987.

[HP98] D. Harel and M. Politi, *Modeling reactive systems with statecharts: The statemate approach*, McGraw-Hill, Inc., New York, NY, USA, 1998.

[HS91] W. A. Halang and A.D. Stoyenko, *Constructing predictable real time systems*, Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[HVF+05] J. Huang, J. Voeten, O. Florescu, P. Putten, and H. Corporaal, *Predictability in real-time system development*, Advances in Design and Specification Languages for SoCs (2005), 123–139.

[ITU] ITU-T , *Recommendation z, specification and description language*.

[Joe00] O. Joel, *Discrete analysis of continuous behaviour in real-time concurrent systems*, Tech. report, Oxford, UK, UK, 2000.

[JOGKL04] D. S. JOVANOVIC, B. ORLIC, and J. F. BROENINK G. K. LIET, *gcsp: A graphical tool for designing csp systems*, Proceedings of Communicating Process Architectures September 2004., 2004.

[Jon90] C. B. Jones, *Systematic software development using vdm (second edition*, 1990.

[KDC96] R. P. Karl, L. Daniel, and K. C. Chan, *Extending statecharts with duration*, COMPSAC '96: Proceedings of the 20th Conference on Computer Software and Applications (Washington, DC, USA), IEEE Computer Society, 1996, p. 246.

[Kop97] H. Kopetz, *Real-time systems. design principles for distributed embedded applications.*, Kluwe academic, 1997.

[Kro87] F. Kroger, *Temporal logic of programs*, Springer-Verlag New York, Inc., New York, NY, USA, 1987.

[Kru95] P. Kruchten, *The 4+1 view model of architecture*, IEEE Softw. **12** (1995), no. 6, 42–50.

[LHH03] M. Lee, T. Hwang, and S-Y Huang, *Decomposition of extended finite state machine for low power design*, DATE '03: Proceedings of the conference on Design, Automation and Test in Europe (Washington, DC, USA), IEEE Computer Society, 2003, p. 11152.

[LJ97] Z. Liu and M. Joseph, *Formalzing real-time scheduling as program refinement*, Proceedings of Transformation-Based Reactive Systems Development, ARTS'97, Lecture Notes in Computer Science 1231, Springer, 1997, pp. 294–309.

[LRA02] W. J. Lloyd, M. B. Rosson, and J. D. Arthur, *Effectiveness of elicitation techniques in distributed requirements engineering*, RE '02: Proceedings of the 10th Anniversary IEEE Joint International Conference on Requirements Engineering (Washington, DC, USA), IEEE Computer Society, 2002, pp. 311–318.

[LRJ01] Z. Liu, L. Le Rh, and M. Joseph, *Verification, refinement and scheduling of real-time programs*, Theoretical Computer Science **253** (2001), 15–2.

[LY03] Q. Li and C. Yao, *Real-time concepts for embedded systems*, CMP Books, 2003.

[MCB07] L.E. Mendoza, M.I. Capel, and K. Benghazi, *Checking behavioural consistency of UML–RT models through trace–based semantics*, Proc. 9th International Conference on Enterprise Information Systems (ICEIS 2007) **3** (2007), 205–211.

[MGD01] J. L. Medina, M. Gonzfilez, and J. M. Drake, *Mast real-time view: A graphic uml tool for modeling object-oriented real-time systems*, In the 22nd IEEE Real-Time Systems Symposium (RTSS'01, 2001, pp. 245–256.

[Mil89] R. Milner, *Communication and concurrency*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[Mil99] ———, *Communicating and mobile systems: the π-calculus*, Cambridge University Press, New York, NY, USA, 1999.

[MJAJ02] O. Mosbahi, L. Jemni, S. Ben Ahmed, and J. Jaray, *A specification and validation technique based on statemate and fnlog*, ICFEM '02: Proceedings of the 4th International Conference on Formal Engineering Methods (London, UK), Springer-Verlag, 2002, pp. 216–220.

[MP92] Z. Manna and A. Pnueli, *The temporal logic of reactive and concurrent systems*, Springer-Verlag New York, Inc., New York, USA, 1992.

[MPW92] R. Milner, J. Parrow, and D. Walker, *A calculus of mobile processes, i*, Inf. Comput. **100** (1992), no. 1, 1–40.

[MT90] F. Moller and C. Tofts, *A temporal calculus of communicating systems*, CONCUR '90: Proceedings on Theories of concurrency : unification and extension (New York, NY, USA), Springer-Verlag New York, Inc., 1990, pp. 401–415.

[NJ94] X.Ñicollin and J.Sifakis, *The algebra of timed processes atp: theory and application*, Information and Computation **114** (1994).

[OMG02] OMG, *Uml profile for modeling quality of service and fault tolerance characteristics and mechanisms*, Object Management Group, Massachusetts, USA, 2002.

[OMG03] ———, *Uml profile for schedulability, performance, and time specification*, Object Management Group, Massachusetts, USA, 2003.

[OMG04a] ———, *UML specification infrastructure - version 2.0*, Object Management Group, Massachusetts, USA, 2004.

[OMG04b] ———, *Uml superstructure specification – version 2.0*, Object Management Group, Massachusetts, USA, 2004.

[OMG05] ———, *UML profile for schedulability, performance, and time specification – version 1.1*, Object Management Group, Massachusetts, USA, 2005.

[OMG07] ———, *UML superstructure specification – version 2.1.1*, Object Management Group, Massachusetts, USA, 2007.

[Par79] D. L. Parnas, *On the criteria to be used in decomposing systems into modules*, 139–150.

[PBGN99] C. E. Pereira, L. B. Becker, M. Gergeleit, and E.Ñett, *An integrated environment for the complete development cycle of an object-oriented distributed real-time system*, ISORC '99: Proceedings of the 2nd IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (Washington, DC, USA), IEEE Computer Society, 1999, p. 165.

[PBO07] R. F. Paige, P. J. Brooke, and J. S. Ostroff, *Metamodel-based model conformance and multiview consistency checking*, ACM Trans. Softw. Eng. Methodol. **16** (2007), no. 3, 11.

[Pnu77a] A. Pnueli, *The temporal logic of programs*, FOCS, 1977, pp. 46–57.

[Pnu77b] ———, *The temporal logic of programs*, SFCS '77: Proceedings of the 18th Annual Symposium on Foundations of Computer Science (sfcs 1977) (Washington, DC, USA), IEEE Computer Society, 1977, pp. 46–57.

[Poo03] J. M. Poole, *Common warehouse metamodel developer's guide*, John Wiley & Sons, Inc., New York, NY, USA, 2003.

[RHB97] A. W. Roscoe, C. A. R. Hoare, and R. Bird, *The theory and practice of concurrency*, Prentice Hall PTR, Upper Saddle River, NJ, USA, 1997.

[RR88] G. M. Reed and A. W. Roscoe, *A timed model for communicating sequential processes*, Theor. Comput. Sci. **58** (1988), no. 1-3, 249–261.

[SBL94] S.Aujla, T. Bryant, and L.Semmens, *Applying formal methods within structured development*, Journal on Selected Areas in Communications **12** (1994), no. 12, 258 – 26467.

[Sch89] S. Schneider, *Correctness and communication in real-time systems (tcsp)*, Ph.D. thesis, 1989.

[Sch00] ———, *Concurrent and real–time systems – the CSP approach*, John Wiley & Sons, Ltd., Chichester, England, 2000.

[Sel03] B. Selic, *Modeling quality of service with uml: how quantity changes quality*, 189–204.

[SGW94] B. Selic, G. Gullekson, and P. Ward, *Real–time object–oriented modeling*, John Wiley & Sons, Ltd., New York, 1994.

[Sha04] R. Sharpe, *Formal methods start to add up again*, Computing (2004).

[SM96] S. Shlaer and S. Mellor, *The shlaer-mellor method.*, 1996.

[Spi92] J. M. Spivey, *The z notation: a reference manual*, Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK, 1992.

[SR98] B. Selic and J. Rumbaugh, *Uml for modeling complex real–time systems*, ObjecTime Technical Report, ObjecTime, New York, 1998.

[SRS98] J. A. Stankovic, K. Ramamritham, and M. Spuri, *Deadline scheduling for real-time systems: Edf and related algorithms*, Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[SWC02] A. Sampaio, J. Woodcock, and A. Cavalcanti, *Refinement in circus*, FME '02: Proceedings of the International Symposium of Formal Methods Europe on Formal Methods - Getting IT Right (London, UK), Springer-Verlag, 2002, pp. 451–470.

[TJ] A. D. Toro and B. B. Jiménez, *Metodología para la elicitación de requisitos de sistemas software.*

[Tov99] E. Tovar, *Contribution for the worst-case response time analysis of real-time sporadic traffic*, in WorldFIP networks. Euromicro Conference on Real-Time Systems RTS'99 (Work-inprogress, 1999.

[Ver94] P. Verssimo, *Ordering and timeliness requirements of dependable real-time programs*, Journal of Real-Time Systems, Kluwer Eds **7** (1994), 14–94.

[Ž94] John J. Žic, *Time-constrained buffer specifications in csp + t and timed csp*, ACM Trans. Program. Lang. Syst. **16** (1994), no. 6, 1661–1674.

[WK99] J. Warmer and A. Kleppe, *The object constraint language: precise modeling with uml*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[Yi90] W. Yi, *Real-time behaviour of asynchronous agents*, CONCUR '90: Proceedings on Theories of concurrency : unification and extension (New York, NY, USA), Springer-Verlag New York, Inc., 1990, pp. 502–520.

[YW03] H. Yang and M. Ward, *Successful evolution of software systems*, Artech House, Inc., Norwood, MA, USA, 2003.

[Zic91] J. Zic, *CSP+T: a formalism for describing real–time systems*, Ph.D. thesis, University of Sydney, Basser Department of Computer Science, July 1991.

[ZLGS] H. Zbigniew, K. Ludwik, R. Gianna, and J. L. Sourrouille.