UNIVERSIDAD DE CANTABRIA

DEPARTAMENTO DE TECNOLOGÍA ELECTRÓNICA
INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

Metodología de Modelo Único para el Diseño de Sistemas Empotrados Heterogéneos basada en UML/MARTE

Memoria presentada para optar al título de
DOCTOR EN CIENCIAS, TECNOLOGÍA Y COMPUTACIÓN
POR LA UNIVERSIDAD DE CANTABRIA

por Pablo Peñil del Campo,
Licenciado en Ciencias Físicas e Ingeniero de Telecomunicación

Director: Héctor Posadas Cobo

Santander, Mayo 2017

Agradecimientos

No voy a dedicar esta tesis a nadie, porque quizá no sea lo suficientemente buena para ello. Lo sí voy a dedicar es el esfuerzo, la energía, las horas de trabajo, el tesón y la ilusión puestos en el desarrollo de este trabajo; todo eso es de lo único que sí estoy seguro.
Una vez dijo Freddie Mercury que toda persona lleva una mochila sobre su espalda. Cada uno la va llenando poco a poco con las piedras que se va encontrando por el camino, sus fracasos, sus victorias, sus miedos e ilusiones o sus obsesiones... En mi caso, mi piedra más pesada es una obsesión: mis padres. Supongo que es el precio que hay que pagar por tener memoria, por tener siempre presente todos los sacrificios que tuvieron que hacer para que yo, su único hijo, tuviera más oportunidades de las que ellos tuvieron. Tantos malos momentos vividos, tantos desprecios e infamias soportadas, tanta mierda tragada y procesada, con el único fin de seguir adelante, de poner pie firme, lambdas al frente, decir eso de "no nos rendimos, esto es un tercio español", aguantando todo y a todos, bailando "ropes-a-dope", solo y únicamente, por mí. Es una gran responsabilidad, a veces pesa en exceso, tanto que te sientes desfallecer. Pero entonces recuerdas, tragas saliva, y das un nuevo paso en el camino del samurái, aquel que te exige lealtad a uno mismo y a lo que uno cree; y a aquellos que por su propia elección y libertad, han elegido acompañarte, de una u otra manera.
Primero he de comenzar con las ausencias que la vida se va cobrando y que hacen que toda victoria sea digna del gran Pirro. A mi padre, luchó contra el muro del tiempo para darme el Dorado; un hombre debería poder disfrutar de los frutos de sus sacrificios y poder ver en quien se está convirtiendo su hijo. A mi abuela, una segunda madre que me enseñó el valor del perdón, la humildad y el respeto; aunque a veces me cueste mucho poder seguir el camino que me marcó.
Mi madre es la alegría hecha vida, una mujer de la vieja escuela, estoica, que afronta las dificultades en silencio, apretando el puño y nunca dando un paso atrás. Digna hija de mi tierra, donde la única manera de vivir es trabajando sin descanso, esforzándose al máximo y nunca dejar de perseverar. Y todo con una sonrisa. Tantas lecciones dadas y tantas por impartir.
A mis amigos (para ellos, esta palabra cada día se queda más corta), Gustavo y Patricia; siempre partiéndonos la caja, quizá porque sabemos muy bien lo que vale disfrutar de los buenos momentos cuando vienen, ya que sabemos demasiado bien cuánto pesan los malos momentos cuando les da por aparecer. Quiero hacer mención especial a su Inés, una sonrisa cautivadora; me tiene loco.
A Mario, mi querido Oddball y su "¿por qué no comentas lo bonito que es todo?" muchas veces tan necesario; sigo pensando que Ramos está muy sobrevalorado, hoy más que nunca. Y Karim es Dios.
A Marián, Nieto y su hija Lucía, una familia que siempre tienen una sonrisa y una mano amiga para mí; no hay mejor regalo.
A Carlos, Verónica, Alicia y Elsa. La vida nos ha separado pero siempre seguiremos unidos por los buenos recuerdos.
Al Fer y Leire, mi matrimonio interracial favorito, siempre agradecidos por su amistad, comprensión y toneladas de paciencia. Y ahora con Alba, en ella está el futuro (siendo racinguista, por supuesto).
A Paz y Carlos, gracias por esas tardes de toros en la Plaza de Cuatro Caminos; el concierto de Jean-Michel Jarre en Santo Toribio va a ser espectacular.
A Evaristo por su excelso conocimiento de la tauromaquia y su forma de trasmitírmela.

A los vecinos de la cuarta planta.
A la tropa de solares (donde cuatro huevos son dos pares), compuesta por el gran Marce y el no menos grande Fiyo (aunque con su aspecto los disimule muy bien), cuantos momentos tan increíbles, cuantas risas vividas, y cuantas por vivir.
A los viejos camaradas de la vieja ratonera: los hermanos Díaz, Luis y Alvarito; al final tanto resistirse para caer rendidos ante la convivencia amancebada, vosotros antes molabais. Alejandro, otro que ya se asoma al precipicio de la vida en pecado, pero resiste como un gatito; poco le queda también a este. Una pena.
A Javi y Juanito Escalador; aún nos queda muchos vermuts por La Latina, muchos picos que andar (aunque en eso, Juan siempre fue el experto).
A Héctor (mi director de tesis) por su paciencia, correcciones y sugerencias que, sin duda, han mejorado la primera versión de esta tesis. También quiero agradecer a Vanesa la prisa que le metía para que se diera prisa en las correcciones.
A la gente increíble que he tenido la suerte de conocer en Madrid y que me han permitido formar parte de sus vidas:

- Martín: FPI conseguida, la siguiente estación, el infinito.
- Alba mi compañera de la montaña, cuánto nos unió cuando casi entregamos la geta al señor bajo aquella ventisca, que frío hacía. Siempre una delicia compartir momentos contigo.
- Cris: entre creer y el crear solo hay una letra de distancia; cree en tus sueños y los crearás. Nunca llegaré a entenderte completamente aunque, normalmente, las cosas son más sencillas de lo que parecen.
- Juan, el dulzor caribeño. Ya es hora que te centres y organices tu vida y tomes el timón de ella, porque si no será ella quién se organice, y eso siempre es malo: es mejor tener un mal plan que no tener plan. Mucha suerte.
- Luis, mi compañero de laboratorio y mi capitán en esta aventura que es el LST. Y ¡Hala Madrid!
- Jorge y Cris. Es una gozada y una maravilla veros juntos. Después del susto de estas navidades, me alegra que todo haya vuelta a su estado normal. Jorge, se acaba nuestro viaje juntos, ha sido una suerte y una privilegio compartir tantas cosas. Muchas gracias y mucha suerte.

También quiero agradecer a todas aquellas personas con las que he compartido trabajo, publicaciones, seminarios… Sin ellos, esta tesis tampoco se podría haber realizado. Mención especial a Julio Medina que durante mis primeros pasos (y los del medio) con MARTE me brindó su tiempo para aclarar todas mis dudas y aportarme sus valiosas opiniones.
A El Marquesado, mi trozuco de Cantabria que tanto amo; un nuevo proyecto apasionante está a punto de comenzar.
A Freddie Mercury, “Love of Life, Singer of Songs”, una constante inspiración en cómo afrontar la vida, buscando siempre la felicidad, sin importar lo que otros piensan o digan.
A la música dance de los 90 (Sash!, Gigi D’Agostino, 2 Unlimited…), sin sus temazos este documento no hubiera sido posible terminarlo.

GRACIAS.

“He visto cosas que vosotros no creeríais: atacar naves en llamas más allá de Orión, he visto rayos C brillar en la oscuridad cerca de la puerta de Tanhauser... todos esos momentos se perderán en el tiempo, como lágrimas en la lluvia.” Nexus-6.

“Imposible es sólo una palabra que usan los hombres débiles para vivir fácilmente en el mundo que se les dio, sin atreverse a explorar el poder que tienen para cambiarlo. Imposible no es un hecho, es una opinión. Imposible no es una declaración, es un reto. Imposible es potencial. Imposible es temporal, Imposible no es nada”. Muhammad Ali.

"Soy orgulloso, y lo seré aunque me estrelle". Cyrano de Bergerac.

“Si realmente quieres hacer algo, encontrarás la manera. Si no, encontrarás la excusa”. Dr. Gregory House.

“No hables de futuro, es una ilusión, cuando el rock’n’roll, conquistó mi corazón”. Loquillo.

Contenido

Índice

Lista de Figuras

Índice

Índice

Lista de Tablas

Lista de Acrónimos

- AADL: Architecture Analysis and Design Language
- API: Application Programming Interface
- BVA: Bounsdary-Value Analysis
- CBSE: Component-Based Software Engineering
- CCSL: Clock Constraint Specification Language
- COMPLEX: COdesign and Power Management in PLatform-Based Design Space EXploration
- CONTREX: Design of embedded mixed-criticality CONTRol systems under consideration of EXtra-functional properties
- CPU: Central Processing Unit
- CSP: Communicating Sequential Processes
- DE: Discrete-Event
- DREAMS: Dynamically Reconfigurable Embedded Platforms for Networked Context-Aware Multimedia Systems
- DSE: Design Space Exploration
- DSP: Digital Signal Processing
- EFR: Enhanced Full Rate
- ESL: Electronic, System Level
- FIFO: First In Firts Out
- ForSyDe: Formal System Design
- FPGA: Field Programming Gate Array
- GPP: General Purpose Processors
- GPU: Graphics Processing Unit
- HetSC: Heterogeneous specifications in SystemC
- IP: Internet Protocol
- JDT: Java Development Toolkit
- LDPC: Low-Density Parity Check
- MARTE: Modeling and Analysis of Real-Time and Embedded Systems
- MAST: Modeling and Analysis Suite for Real-Time Applications
- MCAPI: Multicore Communications API
- MDD: Model-Driven Design
- MDE: Model Driven Engineering
- MoC: Model of Computation
- MOET: Maximum Observable Execution Time
- MOFM2T: Model To Text Transformation Language
- MPSoC : Multi-Systems-on-Chip

- MTL: Model to Text Language
- OpenMP: Open Multi-Processing
- OSI: Open System Interconnection
- PDM: Platform Description Model
- PHARAON: Parallel and Heterogeneous Architecture for Real-time ApplicatiONs
- PIM: Platform Independent Model
- PLD: Programable Logic Device
- POSIX: Portable Operating System Interface
- PSM: Platform Description Model
- RMI: Remote Method Invocation
- SCoPE: SoC Co-simulation and Performance Estimation in SystemC
- SDF: Synchronous-Data-Flow
- SoC: System on Chip
- SATURN: SysML bAsed modeling, architecTUre exploRation, simulation and syNthesis for complex embedded systems
- TDMA: Time Division Multiple Access
- TLM: Transaction-Level Modeling
- TOISE: Trusted Computing for European Embedded Systems
- UML: Unified Modeling Language
- UTP: UML Testing Profile
- VHDL: VHSIC+HDL
    - VHSIC: Very High Speed Integrated Circuit
    - HDL Hardware Description Language
- VIPPE: Virtual Platform Parallel Performance Evaluation
- VSL: Value Specification Language
- WSN: Wireless Sensor Networks
- XML: eXtensible Markup Language
- YAW: Yet Another Waveform

Índice

Capítulo I

Introducción

Todo proceso de diseño presenta retos y riesgos inherentes a la propia complejidad de la tarea que se está realizando. Así, en los procesos de diseño de sistemas electrónicos, los ingenieros han de abordar un conjunto más o menos grande de tareas complejas, en función del sistema que tengan que desarrollar. La manera de abordar esta tarea se ve facilitada por la experiencia y conocimiento del propio diseñador.

Como consecuencia de esta complejidad, es muy común que durante dicho proceso de diseño se deban utilizar varios tipos de herramientas, con el fin de poder abordar la realización de cada una de las tareas concretas que componen dicho proceso de diseño. Este hecho implica que los ingenieros tienen la necesidad de conocer, e incluso dominar muchas herramientas, lo que puede dar lugar a una serie de dificultades colaterales al diseño del sistema en sí, que han de ser neutralizadas. En primer lugar, el uso de distintas herramientas puede generar una potencial pérdida de tiempo en la migración de los datos de unas herramientas a otras. En ocasiones para realizar esta migración se ha de duplicar la información, ya que no hay forma de intercambiar información entre herramientas directamente. En segundo lugar, para saber manejarlas y sacar su máximo potencial, el ingeniero ha de utilizar una gran cantidad de tiempo en su formación y en la adquisición de experiencia, lo que no siempre es posible.

El problema de necesitar la especialización de cada uno de los diseñadores en un gran número de herramientas se puede reducir con la creación de grupos de trabajo. De esta forma, cada uno de estos grupos se puede responsabilizar de una o varias tareas del proceso de diseño, especializándose en las herramientas asociadas a ellas. Sin embargo, esto no resuelve todos los problemas. El problema de migrar la información asociada a los diseños de unas herramientas a otras aún persiste. Además, este problema se puede agravar con la generación de grupos de trabajo, ya que ésta compartimentación del trabajo requiere de mucha comunicación y coordinación entre los distintos grupos, con el fin de evitar desajustes, problemas de integración, redundancias, malos entendidos, etc., sobre todo si son necesarias vueltas atrás en el proceso de diseño.

Ciñéndonos al contexto de esta tesis, el planteamiento propuesto en los párrafos anteriores se vuelve especialmente importante, ya que el proceso de diseño de sistemas empotrados requiere de la interrelación de un gran número de disciplinas (especificación, análisis y verificación, implementación…) como consecuencia de su complejidad. Además, el diseñador o grupo de diseño puede abordar la tarea de desarrollo del sistema empotrado de múltiples formas, dependiendo de los objetivos de su proyecto. En algunos casos, es posible que la plataforma destino venga dada y se quiera desarrollar únicamente la funcionalidad, considerando las particularidades necesarias para ejecutarla en dicha plataforma; en otros casos, puede que se quieran explorar las potenciales plataformas

disponibles para ejecutar en ellas una funcionalidad, adaptando su implementación a las características de cada plataforma; en otros casos es posibles que se desee el diseño de una plataforma específica para el sistema; etc…

Para superar estas dificultades, en el diseño de los sistemas empotrados una de las alternativas más eficaces es la de comenzar por abordar la especificación de la funcionalidad, centrándose más en qué se quiere hacer que en cómo ha de hacerse. A partir de ahí, esta especificación ha de evolucionar para transformarse en un modelo que pueda incluir también detalles de cómo se ha de ejecutar dicha funcionalidad. Para este fin, el diseñador tendrá que definir tanto la plataforma hardware donde dicha funcionalidad sea ejecutada como la forma en que dicha plataforma será usada para ejecutar la funcionalidad.

Durante esta tarea, se deberán realizar una serie de análisis que permitan ir estudiando las decisiones tomadas en relación a la funcionalidad, a la plataforma, y a la utilización de dicha plataforma como recurso para la ejecución de la funcionalidad. De esta manera, el diseñador tendrá la posibilidad de analizar lo correcto de las decisiones que ha ido tomando a lo largo del proceso de diseño y explorando otras que le permitan obtener la configuración final más óptima de su sistema, de acuerdo con las especificaciones de diseño a cumplir.

El problema es que la cantidad de posibles escenarios que se pueden encontrar en el diseño de sistemas empotrados es muy amplia, con lo que la complejidad para realizar todas estas actividades es notable. Con estas premisas se hace necesario el desarrollo de soluciones integrales, que permitan el manejo de múltiples herramientas, y que nos doten de entornos suficientemente completos y flexibles donde los diferentes aspectos de un sistema empotrado puedan ser especificados, analizados, optimizados y verificados.

Para considerar todos estos aspectos, se ha de cubrir tanto la funcionalidad, como la plataforma donde dicha funcionalidad será mapeada para su ejecución. Además de esto, se ha de permitir realizar el conjunto de escenarios de análisis donde las decisiones de diseño tomadas serán estudiadas, a través de una serie de etapas y mediante un conjunto de herramientas complejo y diverso.

1. Elementos a considerar en el diseño de sistemas empotrados

1.1 Aplicación

Todo sistema empotrado tiene como objetivo ser capaz de desarrollar las actividades necesarias para cumplir las funciones para las que es diseñado. Para ello, los sistemas empotrados deben incluir una o varias aplicaciones capaces de realizar las funcionalidades requeridas. Así, durante el proceso de diseño se deberá abordar el desarrollo de dichas aplicaciones, de tal forma que los resultados obtenidos sean lo mejor posibles en función de los recursos disponibles. Esto significa que el proceso de diseño deberá cubrir detalles tales como la estructura de la aplicación, su codificación, su rendimiento, su adaptación a la infraestructura disponible, etc.

La forma en la cual esta funcionalidad es implementada tiene mucha relevancia, en relación al impacto sobre el rendimiento y al esfuerzo necesario para su desarrollo. Una primera aproximación puede ser verla como un todo, sin ninguna estructura, de tal forma que el algoritmo implementado sea una única unidad. Sin embargo, esta estrategia resulta muy ineficiente en términos de manteniendo del código, reusabilidad del mismo, optimación a la hora de adaptarlo a nuevos proyectos y, sobre todo, a la imposibilidad de obtener provecho de la heterogeneidad de las plataformas actuales.

Otra estrategia de diseño más eficiente consiste en dividir esta funcionalidad en unidades jerarquizadas, con el fin de organizar el algoritmo como un conjunto de unidades funcionales que interactúan; modularizándola. En el contexto de la ingeniería software, la metodología de desarrollo software CBSE ("Component-Based Software Engineering") [1] identifica esas unidades funcionales como componentes.

El principio de estructurar la funcionalidad en base a componentes tiene grandes ventajas a la hora de ser aplicada al diseño de sistemas empotrados.

En las plataformas actuales es bastante frecuente tener múltiples recursos de cómputo. Este hecho hace que la división de la aplicación permita un mapeo más flexible, aprovechando el incremento de recursos de cómputo en las plataformas actuales. Así, el diseñador debe tener en cuenta la concurrencia como elemento clave para la optimización de los sistemas. La concurrencia permite la ejecución simultánea de diferentes funcionalidades, optimizando el uso de los recursos de la plataforma. Sin embargo, esta concurrencia añade nueva complejidad al sistema, en términos de riesgos de inconsistencia de datos, carreras críticas, inter-bloqueos… La división de la funcionalidad

en componentes, donde los flujos de control y las comunicaciones están claramente identificados, facilita el análisis y resolución de estos problemas.

Un buen ejemplo es un sistema de adquisición y procesado de imágenes. En él, estructurar el sistema en dos partes, una de adquisición y otra de procesado, hace que ambas tareas puedan ser ejecutadas en paralelo. De esta forma, se podrá adquirir una nueva imagen mientras la anterior está siendo procesada, reduciendo el tiempo de espera entre capturas. Si no, al esperar a adquirir una imagen hasta que la anterior se haya procesado completamente, se consumirá un mayor tiempo entre adquisición de imágenes, reduciendo el rendimiento del sistema. De esta forma, se puede mejorar el rendimiento nuestro sistema aprovechando el paralelismo de dichas plataformas.

Así pues, una decisión complementaria a la partición en componentes de la funcionalidad hace referencia al cómo esos componentes se van a comunicar, qué propiedades van a ir asociadas a cada interconexión de dichos componentes. Decisiones como utilizar llamadas síncronas o asíncronas, o integrar canales con memoria o prioridades son ejemplos de lo que se puede ser considerado durante el proceso de diseño.

También, gracias a la división en componentes, el trabajo de desarrollo se puede organizar en grupos de trabajo, una vez que se han definido las interfaces entre los componentes. Esta división también beneficia el mantenimiento y la optimización del sistema, al ser posible desarrollar nuevas versiones de los componentes sin necesidad de modificar el sistema completo, de forma que sea posible arreglar fallos o mejorar el rendimiento en relación a nuevos requisitos funcionales o nuevos avances en la tecnología.

De la misma forma, la división del sistema en componentes da flexibilidad a las posibilidades en cuanto al origen de los códigos utilizados para cada componente, ya que la codificación de la funcionalidad puede venir heredada de otro proyecto previo, venir de un ente externo o formar parte del proceso de diseño. Así mismo, la posibilidad de reutilizar componentes facilita el diseño de nuevos sistemas, aprovechando la experiencia de los anteriores al tener una versión de los componentes optimizada y probada. Por ello, la estructuración en componentes permite reutilizar los mismos componentes en diferentes proyectos de desarrollo, ahorrando tiempo y esfuerzo.

1.2 Plataforma: recursos HW/SW

Las plataformas HW han experimentado un gran desarrollo en las últimas décadas, pasando de sistemas mono-procesador a la amplia gama de plataformas multi-

procesadoras que se pueden encontrar hoy en día. Así mismo, también se ha producido una diversificación de los tipos procesadores, con objeto de desarrollar soluciones específicas para un determinado tipo de problemas. En consecuencia, estas nuevas plataformas incluyen recursos de procesamiento de diferentes tipos, más allá de los procesadores de propósito general ("General Purpose Processors", GPP), como DSPs ("Digital Signal Processing") orientados al procesamiento numérico a alta velocidad, GPUs ("Graphics Processing Unit") orientadas al procesamiento gráfico, u otro tipo de coprocesadores que permiten aligerar la carga de los procesadores principales desarrollando algún tipo de operación concreta, como operaciones vectoriales o de punto flotante.

También hay que considerar que las plataformas HW pueden añadir a los recursos anteriores HW especializado o FPGAs ("Field Programming Gate Array"), para la integración de HW específico que permita optimizar el funcionamiento del sistema.

Como se indicó anteriormente, la partición en componentes de la funcionalidad permite un mapeo más flexible en la plataforma destino. Además, el paralelismo que proporcionan que los distintos recursos de dichas plataformas puedan ser explotados de forma más sencilla gracias a la división en componentes. Pero esta complejidad de las plataformas tiene también otra consecuencia a parte de la disposición de múltiples recursos de cómputo: el aumento de la heterogeneidad de tipos de dichos recursos. De esta forma, el binomio componentes-concurrencia facilita también el diseño de sistemas en relación a las plataformas heterogéneas actuales, ya que tener la funcionalidad dividida en componentes permite la asignación directa de esos componentes a dichos recursos y su optimización conforme a las características de los mismos. Así es posible realizar el diseño del sistema aprovechando no solo su número de procesadores, sino la heterogeneidad de los mismos, en la búsqueda de un mejor rendimiento del sistema.

Junto con los recursos HW disponibles, en el desarrollo del sistema también se deben considerar los diferentes recursos SW necesarios y disponibles. Durante el proceso de diseño se han de tener en cuenta qué recursos SW de la plataforma van a ser usados para la implementación de las comunicaciones, cómo se van a implementar los diferentes flujos concurrentes, etc.

Además, se debe considerar si vamos a trabajar con una plataforma local o si tendremos un sistema distribuido, compuesto de múltiples nodos donde la aplicación vaya a ser mapeada, y que presentan sus propios desafíos a la hora de abordar su diseño.

2. Etapas del diseño de sistemas empotrados

Cada decisión tomada durante la fase del diseño de un sistema debe ser estudiada para analizar su corrección y el impacto que tiene sobre las prestaciones del sistema, en relación a los requisitos impuestos a dicho sistema. Esta necesidad de análisis ha de ser entendida no como una única tarea o un único aspecto a analizar del sistema; la gama de propiedades, decisiones, etc. a evaluar del sistema es muy amplia, según los objetivos del diseñador y la etapa de diseño en curso. De esta manera, se podrá establecer un proceso continuo de evaluación y refinamiento del diseño, con el fin de obtener la implementación final del mismo de manera progresiva.

Así, para abordar el diseño de los sistemas empotrados hay que realizar un gran número de tareas que pueden ser representadas por la Figura 1. Esta figura resume lo descrito en la sección anterior.

Como ramas principales del proceso de diseño, el desarrollo de la aplicación y su integración con la plataforma son parte intrínseca del diseño del sistema empotrado. Respecto a la aplicación, su estructura, los mecanismos de comunicación, la concurrencia… son aspectos a considerar. Además, todas estas decisiones han de ser validadas mediante técnica de análisis, que nos reporten información sobre la correcta implementación de la funcionalidad a desarrollar.

En cuanto a las decisiones que se pueden tomar para obtener la plataforma que mejor se adapte a las necesidades del sistema, estas pueden estar relacionadas con el número y tipo de los recursos de cómputo presentes en ella, así como con las propiedades de los diferentes recursos HW presentes en dicha plataforma (tamaños de las memorias, frecuencias de procesadores…), los cuales tienen un impacto directo sobre el futuro rendimiento del sistema.

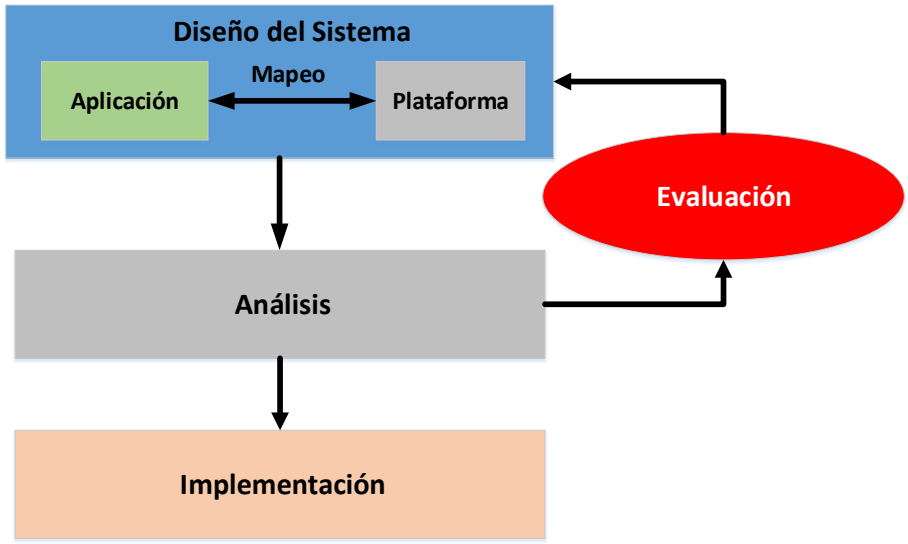


Figura 1 Tareas del diseño de sistema empotrados

Una vez que se ha diseñado la aplicación y la plataforma, se ha de realizar la tarea de mapear la aplicación a los recursos de cómputo disponibles en la plataforma destino.

Con este mapeo, se cierran los principales detalles de la implementación, permitiendo considerar parámetros no funcionales en las actividades de análisis. Con ello, aún a falta de realizar las distintas etapas de refinado y optimización, es posible comenzar a evaluar los resultados que se obtendrán como consecuencia de las decisiones tomadas. En este sentido, se puede analizar no solo lo idóneo del mapeo hecho, sino lo correcto de la elección de la plataforma, o de las propiedades conferidas a la aplicación. Esto puede involucrar una redefinición de la aplicación en términos, por ejemplo, de la concurrencia, o de seleccionar una nueva plataforma que posea, por ejemplo, más procesadores.

En consecuencia, para poder realizar las evaluaciones necesarias, hay que tener en cuenta que hay una gran variedad de variables que afectan directamente al rendimiento del sistema: el impacto que tiene la estructura de concurrencia definida, los recursos HW y SW utilizados, la utilización de esos recursos para ejecutar e implementar la funcionalidad… En este contexto, se hace imprescindible introducir en el entorno de desarrollo diferentes tipos de mecanismo de análisis, con el fin de obtener estimaciones del impacto en el rendimiento del sistema de las decisiones que se van tomando en el proceso de diseño. Por ello, es necesario manejar un conjunto diverso de herramientas en el proceso de diseño.

Además, es necesario poder usar estas herramientas de forma que sea posible realizar los diferentes tipos de análisis con el menor esfuerzo posible, con el fin de poder

probar tantas configuraciones como sea necesario hasta encontrar una solución que se ajuste a las necesidades. Para ello la integración de las distintas herramientas en un mismo entorno de desarrollo global se convierte en una necesidad más que en una recomendación.

3. Propuesta

Para abordar todos los retos que requiere el diseño de los sistemas empotrados, especialmente en el manejo e integración de múltiples etapas y herramientas de refinado y análisis, esta tesis presenta como solución la creación de un entorno de desarrollo integrado donde los diseñadores puedan abordar las tareas de diseño de forma ágil a partir de un modelo de alto nivel común.

Así, el entorno de diseño que se presenta en esta tesis se sustenta en dos pilares:

- El primero de ellos es que toda la descripción y caracterización del sistema se hace mediante un modelo de alto nivel global. En este modelo se podrá capturar toda la información referente al diseño de la funcionalidad, la plataforma, el mapeo de los componentes funcionales a los recursos de la plataforma, los escenarios de análisis, el entorno del sistema, etc. De esta forma, el sistema estará completamente caracterizado. El modelado capturará por tanto toda aquella información que es relevante no solo para su especificación, sino para permitir la realización de todo el conjunto de análisis necesarios para obtener la información a considerar en el proceso de diseño. Además, el modelo también incluirá la información necesaria para poder realizar tanto modelos virtuales como la implementación de prototipos del sistema, permitiendo realizar evaluaciones reales una vez que las primeras etapas del proceso de análisis hayan finalizado.
- El segundo pilar del trabajo es la propia integración del conjunto de herramientas necesarias para que el diseñador pueda realizar las distintas actividades del proceso de diseño de manera ágil, que incluirán especificar, analizar e implementar el sistema a diseñar. Así, mediante un entorno de fácil manejo, se podrá especificar el sistema, realizar diferentes análisis del diseño, realizar procesos de síntesis HW/SW, simulación funcional… Todas estas tareas se podrán ejecutar de forma automática ya que, partiendo de la información capturada en el modelo de alto nivel, las diferentes herramientas podrán ser lanzadas. De esta forma, se facilita la labor del diseñador de manera que tenga a su disposición, de forma fácil, el acceso y la utilización de las herramientas que más le interesen en cada momento de su proceso de desarrollo.

Para desarrollar y demostrar el gran grado de aplicabilidad y las enormes capacidades de la idea propuesta, durante la tesis se ha desarrollado una metodología y un entorno de desarrollo en el que se han integrado diversas herramientas, que se operan recogiendo automáticamente la información capturada en el modelo de alto nivel. Estas herramientas han cubierto análisis tanto estáticos como dinámicos, verificando la funcionalidad y analizando requisitos no funcionales, como tiempos de ejecución, o planificabilidad. Además, se ha trabajado con especificaciones ejecutables de alto nivel, con modelos virtuales y con prototipos reales.

La versatilidad del entorno de diseño también ha de ser entendida desde otra perspectiva: el diseñador ha de tener la suficiente flexibilidad para permitirle trabajar en diferentes escenarios de diseño, en función de los objetivos que quiera cumplir. De esta forma, el entorno desarrollado no tiene como objetivo imponer ningún flujo de diseño estricto, sino servir de soporte a las distintas etapas y herramientas que el diseñador necesite utilizar. En función de su criterio, el propio diseñador establece las etapas que quiere realizar en su proceso de desarrollo, ejecutando las herramientas correspondientes. Por ejemplo, estas pueden abarcar una exploración amplia de variables de diseño, o simplemente, un reducido número de potenciales diseños que, con la generación rápida de prototipos, poder evaluarlos.

3.1 Modelo de Alto Nivel

El modelo único que servirá para especificar el sistema y como entrada común a todas las herramientas cubiertas en esta tesis se creará usando el lenguaje de modelado UML [2]. UML nació como un lenguaje para la especificación de sistemas software orientados a objetos, convirtiéndose en un estándar ampliamente usado para la ingeniería software. Diseñar, documentar, organizar, y gestionar el mantenimiento de proyectos son tareas donde UML es ampliamente usado y valorado.

Por ello, su uso se ha extendido de forma muy relevante. Esta gran versatilidad se debe a que UML tiene una gran capacidad expresiva; posee una gran cantidad de mecanismos capaces de capturar todos los aspectos de un sistema, permitiendo describir tanto la estructura estática como el comportamiento dinámico de un sistema:

- la estructura estática define los objetos que van a componer el sistema;
- el comportamiento dinámico define la historia de los objetos en el tiempo y la comunicación entre objetos para cumplir sus objetivos.

Sin embargo, a pesar de ser una buena base para otros campos, dado que UML está orientado a la ingeniería software, sus capacidades están centradas en ese campo, presentando ciertas limitaciones en su aplicación a otros. Con el fin de ampliar el área de aplicación de UML, tanto su semántica como sus posibilidades expresivas han sido extendidas con nuevos recursos a través de diversos trabajos y estándares.

En particular, para poder ser utilizado como lenguaje de modelado de sistemas electrónicos, UML se ha enriquecido con nuevos mecanismos expresivos que nos permiten añadir información referente a otros aspectos del sistema, como la plataforma HW/SW, mapeos arquitecturales o modelos de análisis. Con este fin, se ha desarrollado el perfil MARTE ("Modeling and Analysis of Real-Time and Embedded Systems") [3]. Este perfil es un estándar desarrollado por la OMG, que permite al diseñador modelar todos los aspectos de un sistema empotrado conforme al estándar (aplicación, plataforma y mapeo) y crear escenarios de análisis que permitan evaluar el modelo del sistema.

En consecuencia, la metodología de modelado de sistemas propuesta en esta tesis se realizará siguiendo la combinación de estándares UML/MARTE (idea original de E. Villar). Con esta combinación se utilizará la capacidad expresiva resultante para introducir en los modelos toda la información que va siendo necesaria para los desarrollos y análisis asociados a las distintas etapas y herramientas que se van cubriendo durante el proceso de diseño.

3.2 Etapas de diseño y análisis

Una vez se va describiendo el sistema en el modelo de alto nivel, este debe ser analizado y refinado, para lo que son necesarias distintas herramientas, en función de las distintas etapas del proceso de diseño. Para garantizar el desarrollo de una solución capaz de soportar herramientas que cubran el mayor número de áreas posibles, se han considerado en la tesis diversos tipos de análisis.

Análisis funcional independiente de plataforma

Un primer tipo de análisis que se ha cubierto en esta tesis es un análisis puramente funcional. En él se considera únicamente la aplicación; independientemente de la plataforma destino donde se mapeará dicha aplicación. Este hecho permite una exploración de la estructura funcional, al poder estudiar el comportamiento del sistema atendiendo a su estructura de componentes, las propiedades de los mecanismos de comunicación o su concurrencia asociada, detectando errores como bloqueos, inanición u otros problemas derivados de una mala implementación de la aplicación.

Así mismo, también se ha considerado la posibilidad de analizar la funcionalidad de forma estática, esto es, a través de mecanismos formales. Si la funcionalidad responde a los principios de ese mecanismo formal, se podrá analizar su comportamiento de forma estricta y aseverar que dicha funcionalidad es correcta por construcción y está libre de riesgos en lo que se refiere a problemas funcionales como bloqueos, etc.

El uso de mecanismos formales también permite la relación entre lenguajes o tipos de descripción o codificación distintos. De esta forma, sería posible capturar un mismo sistema con lenguajes diferentes y, si ambos tienen la misma representación formal, esto implica que ambos modelos contienen las mismas propiedades funcionales y garantizaría la corrección de las distintas evoluciones que puede ir sufriendo el sistema conforme avanza el proceso de implementación. El mecanismo formal sería, por tanto, el garante de que esa transformación es correcta por construcción.

Refinado y optimización dependiente de plataforma

Además de validar la funcionalidad, en el diseño de sistemas empotrados es habitual que se deban considerar otros parámetros no funcionales, como rendimiento o consumo. Sin embargo, para poder analizar estos parámetros no es suficiente con considerar únicamente la funcionalidad, sino que también es necesario considerar la plataforma y así comenzar un proceso de refinado del sistema en conjunto.

En este contexto se requiere de procesos capaces de generar modelos ejecutables o prototipos a partir del modelo abstracto de alto nivel, donde puedan ser evaluados los efectos de la plataforma en la ejecución de la funcionalidad. Estos procesos deberán incluir, a sí mismo, mecanismos de síntesis (SW o HW) que permitan generar los códigos necesarios para que la funcionalidad pueda ser ejecutada conforme a los mapeos arquitecturales elegidos, como se irá viendo en el resto del documento.

A continuación, estos modelos o prototipos podrán ser utilizados para realizar análisis temporales que permitan evaluar los plazos necesarios para la ejecución de las funciones, verificando que se cumplen las restricciones temporales con los recursos disponibles en la plataforma.

Además de esto, el uso de modelos virtuales ejecutables permitirá estudiar el impacto de diversas configuraciones de plataforma, atendiendo a las propiedades de sus elementos, al número de ellos que estén presentes en la misma… En el mismo sentido, se puede analizar el impacto del mapeo de la funcionalidad en dicha plataforma, analizando

si un recurso no tiene capacidad suficiente para ejecutar la funcionalidad asignada, modificando dicho mapeo en la búsqueda de un mejor balanceo del sistema.

En la búsqueda de una mejor configuración del sistema, se puede establecer, por tanto, un proceso de exploración de diseño, considerando como variables las propiedades de la plataforma y el mapeo de la funcionalidad.

En resumen, gran parte del trabajo presentado en esta tesis consiste, pues, en la integración, entorno al modelo UML/MARTE, de un conjunto de herramientas que permite al diseñador ir realizando el conjunto de tareas descritas anteriormente de forma ágil.

Para ello se ha trabajado con los siguientes objetivos:

- Para realizar el modelado completo del sistema se podrá usar a todos aquellos mecanismos expresivos seleccionados en esta metodología y que son aportados tanto UML como MARTE. De la misma forma, se ha de permitir desarrollar, cuando sean imprescindibles, nuevos mecanismos expresivos para poder capturar algún aspecto del sistema que no se pueda hacer con MARTE.
- En segundo lugar, es necesario que el entorno de modelado permita que se integren y se conecten las diferentes herramientas de forma que se pueda transferir la información del modelo a las herramientas de forma automática. De esta forma, el diseñador podrá realizar las diferentes etapas de su diseño de forma rápida y sencilla.
- Como UML/MARTE permite la generación de modelos no ejecutables, se necesario integrar en la infraestructura desarrollada al menos una herramienta que transforme la información capturada en el modelo en algún lenguaje de acción, de tal forma que sea posible realizar la simulación funcional.
- De la misma forma, se necesitan herramientas que permitan realizar la síntesis HW y SW para generar los códigos necesarios para crear los modelos ejecutables o prototipos a analizar.
- Así mismo, será necesario integrar herramientas que permitan realizar los análisis de planificabilidad, exploración del espacio de diseño, etc. a partir de la información capturada en el modelo.

4. Proyectos de Investigación

Esta tesis se ha desarrollado en el contexto de varios proyectos de investigación, en los cuales han participado diferentes entidades nacionales e internacionales. Los proyectos de investigación son:

- **SATURN** (SysML bAsed modeling, architecTUre exploRation, simulation and syNthesis for complex embedded systems). Es un proyecto FP7 que tenía como principal objetivo rellenar el vacío existente entre el modelado y la verificación/síntesis en diseños basados en UML de Sistemas empotrados compuestos de HW y SW, [vv)]. Director principal: E. Villar; presupuesto: 193200€; periodo: 2008-2010.
- **CONTREX** (Design of embedded mixed-criticality CONTRol systems under consideration of EXtra-functional properties). Es un proyecto FP7 que estaba centrado en el diseño de sistema de sistemas, enfocado en aquellos que presenten criticidad mixta, [h)], [ll)], [ww)]. Director principal: E. Villar; presupuesto: 391336; periodo: 2013-2016.
- **COMPLEX** (COdesign and Power Management in PLatform-Based Design Space EXploration). Es un proyecto FP7 que estaba enfocado a desarrollar una metodología de diseño para explorar iterativamente el espacio de diseño de sistemas HW/SW, [xx)]. Director principal: E. Villar; presupuesto: 325588€; periodo: 2009-2013.
- **TOISE** (Trusted Computing for European Embedded Systems). Es un proyecto ENIAC cuyo objetivo era el estudio de soluciones seguras resistentes a manipulaciones, en el contexto de aplicaciones embebidas tales como redes de sensores, [yy)]. Director principal: P. Sánchez; presupuesto: 163861€; periodo: 2011-2013.
- **DREAMS** (Dynamically Reconfigurable Embedded Platforms for Networked Context-Aware Multimedia Systems). Es un proyecto nacional que tenía como objetivo contribuir al desarrollo de métodos y herramientas para sistemas empotrados en red, desarrollando plataformas HW/SW adaptables y seguras, que ejecutarán algoritmos multimedia "inteligentes", adaptados a dicha plataforma, [zz)]. Director principal: P. Sánchez; presupuesto: 141086€; periodo: 2011-2013.
- **PHARAON** (Parallel and Heterogeneous Architecture for Real-time ApplicatiONs). Es un proyecto FP7 que estaba enfocado a la reducción del consumo de energía y mejorar el rendimiento de los sistemas empotrados, proporcionando nuevos paradigmas para la programación de arquitecturas multi-núcleo, [aaa)]. Director principal: E. Villar; presupuesto: 354000€; periodo: 2011-2014.
- **CRAFTERS** (Constraint and Application driven Framework for Tailoring Embedded Real-time Systems). Es un proyecto ARTEMIS, [bbb)]. Director principal: P. Sánchez; presupuesto: 310000€; periodo: 2012-2015.

El hecho de que la labor realizada en esta tesis se haya desarrollado durante estos proyectos implica que este trabajo de tesis se ha ido haciendo de forma colaborativa con

diferentes personas. A lo largo de este documento se ha intentado citar a dichas personas en las diferentes áreas en las que han contribuido.

5. Estructura del documento

El documento de esta tesis se estructura de la siguiente manera. Como se ha mostrado en esta sección, el capítulo primero presenta la introducción a esta tesis, mostrando las líneas maestras de este trabajo. El segundo capítulo muestra el estado del arte en lo que a UML y MARTE se refiere y su aplicación al diseño de sistemas empotrados, así como una breve introducción a los lenguajes y herramientas que se han considerado como parte de la infraestructura desarrollada en esta tesis. En el capítulo tres se muestra en conjunto la solución propuesta en esta tesis: un modelo de alto nivel como solución integral para abordar el diseño de los sistemas empotrados y su aplicación a diversas herramientas. En el capítulo cuatro se describe en más detalle la metodología de modelado UML/MARTE, estructurada en función de los diferentes aspectos del sistema que se pueden capturar y diseñar. En el capítulo cinco se describe cómo se integran las diferentes herramientas que van a dar soporte de uso a la metodología de modelado que se presenta. En el capítulo seis se presentan un conjunto de casos de uso para mostrar cómo se aplican conjuntamente la metodología y las herramientas para realizar el desarrollo de sistemas empotrados. El capítulo siete incluye las conclusiones sobre los resultados de esta tesis y, finalmente, en los capítulos ocho y nueve se incluyen las referencias a los artículos producidos durante esta tesis, y las referencias a otros trabajos, respectivamente.

En cada de las secciones que componen los capítulos *Metodología de Modelo Único* y *Entorno de Desarrollo: Integración de Herramientas* se muestran, al final, los proyectos de investigación en los que dicha sección fue desarrollada y los artículos científicos relacionados con el tema de esa sección.

Capítulo II

Estado del Arte

El estado del arte se va a estructurar entorno a los trabajos previos presentes en la literatura que abordan el modelado de sistemas con UML y MARTE y su aplicación al diseño de sistemas empotrados. Estos trabajos abarcaran todos los aspectos que se presentan en esta tesis, centrándose en cómo el modelo ULM/MARTE es utilizado como elemento central de donde parten las distintas actividades de diseño. Así mismo, se ha intentado cubrir todos los diferentes aspectos del proceso de diseño que se tratan en esta tesis.

Para ello se comenzará con una breve descripción de los aspectos más genéricos de UML y de MARTE, como introducción a estos dos lenguajes, y luego se pasará a analizar los distintos trabajos asociados.

1. UML

UML ("Unified Modeling Language") es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar sistemas software orientados a objetos, como inicial dominio de aplicación. Además de esto, permite capturar decisiones y conocimientos sobre los sistemas que se deben construir. UML está pensado para usarse con todos los métodos de desarrollo, etapas del ciclo de vida, dominios de aplicación y medios.

UML permite capturar la información sobre la estructura estática y el comportamiento dinámico de un sistema. Un sistema se modela como una colección de objetos discretos que interactúan para realizar una cierta actividad. La estructura estática define los tipos de objetos. El comportamiento dinámico define la historia de los objetos en el tiempo y la comunicación entre objetos para cumplir sus objetivos.

UML define una serie de trece diagramas, que se utilizan para modelar diferentes aspectos de un sistema. Como estos diagramas están altamente entrelazados, la información plasmada en ellos es a menudo redundante. Se dispone de tres tipos diferentes de diagramas: los que dan una vista de la estructura del sistema, los que capturan el comportamiento dinámico de los objetos del sistema y los de interacción.

Los primeros son:

- Diagrama de clases: muestra las clases, interfaces, colaboraciones y sus relaciones. Dan una vista estática del proyecto.

- Diagrama de objetos: Es un diagrama de instancias de las clases mostradas en el diagrama de clases. Muestra las instancias y como se relacionan entre ellas. Se da una visión de casos reales.
- Diagrama de componentes: Muestran la organización de los componentes del sistema. Un componente se corresponde con una o varias clases, interfaces o colaboraciones.
- Diagrama de despliegue: Muestra los nodos y sus relaciones. Un nodo es un conjunto de componentes. Se utiliza para reducir la complejidad de los diagramas de clases y componentes de un gran sistema. Sirve como resumen e índice.
- Diagrama de estructura compuesta: muestra la estructura interna de una clase o componente, que pueden incluir la especificación de sus partes internas, de puertos, mediante los cuales las partes interactúan con cada una de las otras, o mediante las cuales, instancias de la clase interactúan con las partes y con el mundo exterior, y también la especificación de conectores entre partes o puertas al exterior.
- Diagrama de paquetes: suministran una descomposición de la jerarquía lógica de un sistema.

Los diagramas que capturan el comportamiento de los objetos del sistema son:

- Diagrama de estados: muestra los estados, eventos, transiciones y actividades de los diferentes objetos. Son útiles en sistemas que reaccionen a eventos.
- Diagrama de actividades: Se utilizan para modelar el funcionamiento del sistema y el flujo de control y/o datos entre objetos.
- Diagrama de casos de uso: Muestran los casos de uso, actores y sus relaciones. Muestra quién puede hacer qué y relaciones existen entre acciones (casos de uso). Son muy importantes para modelar y organizar el comportamiento del sistema.

Finalmente, el tercer conjunto de diagramas son:

- Diagrama global de interacción: son muy similares a los diagramas de actividad. Los diagramas de interacción muestran una secuencia de diagramas de interacción, esto es, es una colección de diagramas de interacción y el orden en que suceden.
- Diagrama de secuencia: muestra cómo los objetos interactúan entre sí y el orden en que se producen esas interacciones. Es importante tener en cuenta que muestran las interacciones para un escenario en particular. Los procesos se representan verticalmente y las interacciones se muestran como flechas

- Diagrama de comunicación: es similar a los diagramas de secuencia, pero el foco está en los mensajes pasados entre objetos
- Diagrama de tiempos: representa el comportamiento de los objetos en un marco de tiempo dado. Si es solo un objeto, el diagrama es directo, pero si hay más de un objeto involucrado, también se pueden usar para mostrar interacciones de objetos durante ese período de tiempo.

2. MARTE

MARTE es un perfil que añade a UML los fundamentos necesarios para el modelado de alto nivel de sistemas empotrados y de tiempo real. Esta extensión pretende proporcionar el soporte necesario para abordar las diferentes etapas de especificación, diseño y verificación/validación de un sistema empotrado.

MARTE proporciona los conceptos necesarios para el modelado detallado de las características de los sistemas empotrados y de tiempo-real, así como de su posterior análisis, por medio de un análisis de planificación de nuestro sistema, o de un análisis de funcionamiento. También proporciona la posibilidad un marco general de análisis que tiene la intención de redefinir o especializar cualquier tipo de análisis.

El perfil MARTE está dividido en tres paquetes (Figura 2), que son "MARTE Foundations", "MARTE Design Model" y "MARTE Analysis Model". Cada uno de ellos describe los conceptos necesarios para su correspondiente ámbito de aplicación.

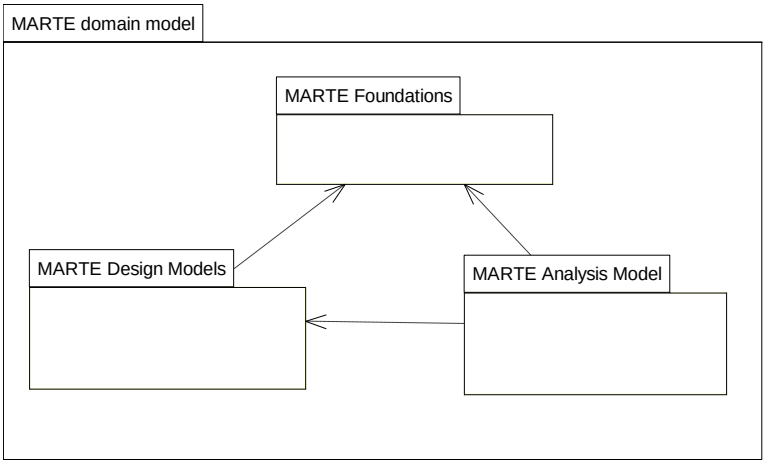


Figura 2 Estructura del Perfil MARTE

2.1 Paquete de Fundamentos

Este paquete define los conceptos fundamentales que forman la base desde donde se desarrollará el perfil. Este paquete está formado por varias secciones.

Elementos centrales ("Core Elements")

Los conceptos presentados en esta sección sirven como base general para la descripción de la mayoría de los elementos del resto de la especificación en le visión del dominio. Está dividido en dos paquetes:

- El paquete de fundamentos proporciona los elementos básicos usados para representar la naturaleza dual descriptor-instancia de cualquier entidad de modelado.
- El paquete "Causality" describe los elementos básicos necesarios para modelar el comportamiento de los elementos de la especificación, así como su semántica en tiempo de ejecución.

Modelado de propiedades no funcionales (NFPs)

Este paquete describe tanto el dominio del modelo como su representación UML para especificar propiedades no funcionales (NFPs). También describe como estas NFPs pueden ser asociadas a los elementos de modelado UML. Esta sección de MARTE proporciona un marco general para anotar los modelos de UML con este tipo de NFPs. También proporciona los constructores necesarios para especificar este tipo de propiedades de una forma detallada.

Modelado de tiempo (Time)

Este capítulo describe un marco general para representar el tiempo y los conceptos y mecanismos que están relacionados con el tiempo, que son apropiados y necesarios para el modelado de los sistemas empotrados y de tiempo real.

Modelado genérico de recursos (GRM)

Esta sección ofrece los conceptos necesarios para modelar una plataforma general para la ejecución de aplicaciones empotradas y de tiempo real.

Modelado de asignación a recursos (Alloc)

Este capítulo proporciona los mecanismos para establecer la asociación entre las aplicaciones y su correspondiente plataforma de ejecución. Un elemento de aplicación puede ser cualquier elemento UML que tenga aspectos estructurales y de comportamiento, con la información más adecuada para modelar una aplicación. Una plataforma de ejecución está representada por un conjunto de recursos conectados, donde cada uno de estos recursos proporciona un servicio para soportar la ejecución de la aplicación. Estos dos ámbitos se modelan separadamente, asociándose después a través de un proceso de asignación.

2.2 Paquete de Diseño

Este paquete cubre los requerimientos del sistema para luego capturar la especificación, el diseño y la implementación que corresponda. Para esto, proporciona principalmente dos familias de conceptos:

- conceptos de alto nivel para modelar características cuantitativas (periodos y plazos), y características cualitativas (concurrencia y comportamiento) de tiempo real.
- conceptos de bajo nivel dedicados a describir los recursos hardware y software usados para la ejecución de la aplicación.

Modelado genérico de componentes (GCM)

Un componente estructurado define una entidad de un sistema, que puede encapsular datos y comportamientos estructurados. Aparte de proporcionar refinamientos sobre este elemento de UML, MARTE especializa principalmente este concepto para soportar los esquemas de comunicación basados en mensajes y en flujos de datos.

Modelado de aplicaciones en alto nivel (HLAM)

Este capítulo proporciona conceptos de modelado de alto nivel para manejar el modelado de características de aplicaciones de tiempo real.

Modelado detallado de recursos (DRM)

El objetivo de este capítulo es proporcionar artefactos de modelado específicos capaces de describir el software, a través del capítulo “Software Resource Modeling”, y

el hardware, a través del capítulo "Hardware Resource Modeling", de nuestro sistema. Especializa los conceptos genéricos ofrecidos en el capítulo "General Resource Modeling" (GRM).

2.3 Paquete de Análisis

Este paquete ofrece conceptos necesarios para el modelar escenarios con los que realizar los diversos análisis de nuestro sistema. El paquete se divide en tres capítulos para abarcar diferentes modos de analizar un sistema.

Modelado genérico para análisis cuantitativos (GQAM)

Este capítulo proporciona los conceptos necesarios para realizar un análisis genérico de nuestro sistema. Aunque el dominio de un análisis puede tener diferente terminología, así como diferentes conceptos y semánticas, comparten conceptos fundamentales que son expresados en este capítulo. Estos conceptos serán especializados en los siguientes capítulos para realizar un análisis concreto del sistema.

Modelado para análisis de planificabilidad (SAM)

Este capítulo está centrado en proporcionar los conceptos necesarios para realizar un análisis de planificabilidad del sistema.

Modelado para análisis de rendimiento (PAM)

Este capítulo describe los conceptos necesarios para los análisis de mejor esfuerzo ("best effort") y de tiempo real blando ("soft real-time") de los sistemas empotrados. Este análisis está determinado por cómo el comportamiento del sistema hace uso de los recursos que tiene.

3. Trabajos previos de modelado de sistemas electrónicos con UML/MARTE

Con el fin de explotar los beneficios de UML como lenguaje de modelado sistemas electrónicos ([34] y [35]) y, más específicamente, para modelar sistemas embebidos y de tiempo real, se ha creado el perfil MARTE, siendo declarado como estándar por la OMG.

Desde su definición, la exploración de las capacidades de modelado de MARTE ha cubierto diferentes etapas del diseño de sistemas electrónicos. Como resultado se han desarrollado trabajos en los que los modelos UML/MARTE han sido usados para actividades como la verificación de la funcionalidad, ya sea a través de modelos ejecutables o de análisis formales, como la síntesis de código (SW o HW) o para el análisis de prestaciones y la exploración del espacio de diseño. Así mismo, también se ha desarrollado algún trabajo donde el mismo modelo se puede usar para

3.1 Generación de modelos ejecutables SystemC desde UML

El primer trabajo a realizar en sistemas construidos mediante la combinación de componentes es garantizar la correcta funcionalidad del conjunto. Para ello, una de las soluciones más extendidas es generar modelos funcionales ejecutables de alto nivel del sistema. Entre estos modelos, han alcanzado especial aceptación en el área de los sistemas empotrados los realizados en el lenguaje SystemC.

A fin de relacionar UML con SystemC y, por tanto, proporcionar semántica ejecutiva a UML, se han propuesto dos principales líneas de investigación. La primera consiste en crear un perfil de SystemC para UML. Mediante los estereotipos y sus correspondientes atributos asociados, se capturan en diagramas UML las especificaciones SystemC [43]. Este enfoque convierte a SystemC tanto en el lenguaje de modelado como en el lenguaje de codificación, al ser UML un mero soporte para capturar gráficamente el sistema. Los modelos UML capturan la semántica de SystemC y, por tanto, son, de facto, especificaciones SystemC. Como resultado estos modelos facilitan el trabajo al principio del proceso de diseño, pero necesitan un esfuerzo adicional conforme avanza el mismo.

Una segunda línea de investigación propone relacionar UML y SystemC estableciendo reglas de mapeo entre el metamodelo UML y los elementos de SystemC. En este caso, UML se utiliza para el modelado de sistemas, mientras que SystemC se utiliza como lenguaje de acción. Por tanto, se establecen unas reglas de mapeo que permiten la generación automática de código ejecutable SystemC desde la información en el modelo UML. En [44] se establece una correspondencia entre los modelos de aplicación capturados con modelos UML y especificaciones SystemC que implementan la plataforma, proponiéndose la definición de unas reglas de transformación que permiten la generación de código semiautomática.

Respecto a la obtención de especificaciones ejecutables SystemC a partir de modelos MARTE, Gaspard2 [40] es el trabajo más destacado.

En esta tesis, esta es la forma de cómo se va a relacionar UML y SystemC; como lenguaje para la generación de especificaciones ejecutables que evalúen, funcionalmente hablando, el sistema.

3.2 Formalización de modelos UML/MARTE

Otra alternativa para analizar la funcionalidad del sistema es utilizar análisis estáticos en lugar de modelos ejecutables. Para ello es habitual crear y estudiar el sistema mediante modelos formales.

UML es un lenguaje de modelado semi-formal, carente de fundamento matemático riguroso. Como consecuencia, se han realizado una gran cantidad de trabajos para cubrir esta carencia de formalismo. El área de trabajo más importante para ello ha consistido en comprender diferentes diagramas UML bajo un cierto formalismo. Desde la versión 2.0 de UML, el hecho de que "las actividades UML han sido rediseñados para utilizar la semántica Petri-Net" [2] generó un gran interés en esta línea de investigación. En [45], los autores establecen una correspondencia entre los elementos de los diagramas de actividad de UML y las redes de Petri, estudiando flujo de datos, control de flujo, nodos de expansión y control de excepciones. En [46] se propone una transformación de los diagramas de actividad a “Object Petri Nets” como validación formal de los modelos para describir el comportamiento del sistema. En [47], se define una semántica formal para diagramas de actividad para soportar la ejecución del flujo de trabajo. Después de eso, los modelos pueden ser analizados para asegurar que estos modelos cumplen requisitos funcionales establecidos.

Otra línea de investigación ha cubierto la definición formal de los diagramas de secuencia UML, como en [48], donde la mayoría de los conceptos de diagramas de secuencia se explican en la semántica formal que proporcionan las redes de Petri (en concreto “Colored High-Level Petri nets”). En [49] los diagramas de secuencia se usan para modelar el comportamiento de las clases que componen el diagrama de clases, usado para modelar la estructura del sistema. Estos diagramas se traducen en un modelo matemático unívoco para detectar inconsistencias en el diseño del sistema.

A fin de proporcionar formalismo a MARTE, F. Mallet et al. han desarrollado el lenguaje formal “Clock Constraint Specification Language” (CCSL) [50]. El objetivo de este lenguaje formal es poder definir modelos de computación (MoCs). Para explorar las capacidades de este lenguaje formal y los conceptos presentes en el capítulo Modelado de Tiempo de MARTE [52], se han publicado diversos trabajos que usan este nuevo lenguaje. En [51] este lenguaje formal se usa para modelar los conceptos del lenguaje

Signal y de las redes de Petri. En [53] diferentes semánticas de comportamiento de AADL ("Architecture Analysis and Desing Language") son modeladas con CCSL. Por lo tanto, a fin de abordar el diseño de distintos ámbitos de un sistema (SW y HW) y para ser capaz de capturar la heterogeneidad de dichos sistemas, es esencial proporcionar los fundamentos formales que permitan el modelado de estos sistemas a través de UML y el perfil de MARTE de una manera adecuada.

En cuanto a la formalización de especificaciones ejecutables en SystemC, el esfuerzo realizado también ha sido importante. En [58] los procesos de SystemC son vistos como máquinas de estado abstractas que consumen y producen datos en cada ciclo delta. La necesidad de concebir el sistema completo en una misma especificación ha dado lugar a la formalización de especificaciones abstractas y heterogéneas en SystemC. En [59] se formalizan especificaciones SystemC que incluyen mapeos a software y hardware. En [60] se formalizan descripciones TLM relacionadas con los sistemas síncronos y en [61] se formalizan especificaciones TLM relativas a sistemas asíncronos. Si bien estos trabajos se centraron en la formalización de los aspectos específicos del diseño de ESL ("Electronic System Level") como la especificación heterogénea, la verificación y la ejecución, sigue sin haber un formalismo común capaz de cubrir a todos ellos.

3.3 Síntesis de código desde UML

Otro de las áreas de estudio, y quizá la que ha concentrado mayor parte del esfuerzo en la integración de UML dentro del diseño de sistemas electrónico, se ha centrado en la síntesis de código. Varios trabajos de investigación sobre síntesis partiendo de modelos UML han sido publicados hasta la fecha. Estos se caracterizan por la creación de modelos basados en máquinas de estado [62] para la posterior generación de un código ejecutable. En [63], se presenta un diseño formal para la síntesis de la lógica de un controlador digital reconfigurable. Mediante máquinas de estado UML, se modelan los superestados concurrentes, permitiendo el mapeo automático de las celdas en la FPGAs.

No obstante, no sólo se han utilizado modelos de máquinas de estado para la síntesis. En [64] se presenta un conjunto de reglas de transformación para realizar la síntesis de código a partir de los diagramas de actividad UML. En esta metodología también se utilizan los diagramas de secuencia, los cuales se usan para definir los patrones de flujo de control, y luego se transforman en diagramas de actividad según otro conjunto diferente de reglas de transformación.

Otro ámbito de investigación relacionado con la síntesis de código se centra en el desarrollo de las comunicaciones HW/SW con metodologías basadas en UML. En [65]

se presenta una solución semiautomática para la generación de infraestructura HW/SW desde modelos UML. Esta solución implementa los controladores software y los adaptadores hardware, utilizando la semántica RMI ("Remote Method Invocation") como entorno para unificar las interfaces de comunicación para todos los componentes HW y SW.

Resumiendo, todas las técnicas anteriores están orientadas a generar modelos completamente fijos, especialmente en su estructura concurrente. Sin embargo, no se considera la exploración de concurrencia, que requiere de un gran esfuerzo al diseñador para encontrar alternativas óptimas de diseño.

Algunas herramientas comerciales ya admiten la generación de código desde UML [66], [67]. No obstante, no producen código dependiente de la plataforma y no admiten diferentes tipos de mapeos arquitecturales.

Además de eso, [68] permite diseñar un sistema basado en FPGAs, soportando la generación automática de descripciones de VHDL a partir del modelo UML/MARTE, estableciendo las reglas de mapeo para traducir elementos de alto nivel en construcciones VHDL, permitiendo la generación de descripciones totalmente sintetizables, incluyendo la estructura y el comportamiento del sistema embebido.

Otros trabajos toman los modelos UML/MARTE como entrada y generan código ejecutable de ellos. En [69] se presenta un flujo de diseño completo que, partiendo de modelos MARTE, se establezca una generación de código, para la implementación de System on Chip (SoCs) reconfigurables dinámicamente.

En [70] se propone un método para la síntesis de interfaces para integración de IP (propiedad intelectual) heterogéneos desde modelos UML. El entorno permite tanto la personalización de los protocolos de interfaz como la generación de la lógica asociada, maximizando así la integración del IP. En [71] se propone una solución para el control de flujo de datos utilizando diferentes mecanismos para manejar hilos de ejecución entre los diferentes núcleos.

3.4 Exploración del espacio de diseño desde UML

También se han abordado distintas tareas de diseño considerando parámetros no funcionales desde UML, como la exploración del espacio de diseño. La metodología Co-Fluent [72] captura la aplicación y la arquitectura HW mediante diagramas de actividad. Los diagramas de actividad de UML se utilizan para especificar los flujos de ejecución

de la aplicación, mientras que MARTE es usado para capturar la plataforma HW. La principal limitación es que exploración requiere de la edición del modelo UML/MARTE y una re-generación del correspondiente ejecutable.

En [73] se presenta una metodología de modelos UML/MARTE que, basándose en hilos de actividad, permita reducir el esfuerzo necesario para capturar el conjunto de mapeos arquitectónicos. Un hilo de actividad es un diagrama actividad donde cada uno de ellos refleja una alternativa de diseño, es decir, un mapeo arquitectónico. Sin embargo, este trabajo no permite la exploración de las propiedades de los diferentes elementos que componen la plataforma HW.

En [74] se presenta una metodología para evaluar soluciones en el particionado HW/SW, identificando los puntos de diseño que cumplen unas restricciones de tiempo dadas. Esta metodología propone una manera de representar, en un conjunto de diagramas UML, todas las combinaciones posibles de las configuraciones del sistema. Mediante la anotación con MARTE de las propiedades no funcionales y mediante la aplicación de análisis de planificabilidad, el espacio de diseño se limita a aquellos puntos que cumplan requisitos temporales. Sin embargo, esta metodología no proporciona soluciones óptimas, ni se apoya en tecnologías automatizadas para la estimación de métricas de rendimiento.

Koski [75] proporciona una interfaz basada en UML para describir la aplicación y la plataforma, añadiendo restricciones de diseño, incluyendo limitaciones a los potenciales mapeos. El entorno también incluye la posibilidad de realizar una generación de código que produce un modelo ejecutable para validación funcional.

3.5 Modelado UML de redes de sensores

Algunos trabajos han propuesto entornos de simulación basados en UML, para modelar y analizar sistemas distribuidos en varios nodos de ejecución, como, por ejemplo, redes de sensores.

En [76], UML se aplica al modelo de redes inalámbricas. En [77] los autores presentan un entorno para modelar y analizar el rendimiento del software de las redes de sensores basado en aplicaciones NesC (lenguaje de programación basado en componentes y en eventos, para la plataforma TinyOS). En [78], se presenta una metodología que, basada en diagramas de secuencia con notaciones MARTE de restricciones temporales, generar, a partir de estos diagramas, una especificación SystemC/TLM para la simulación de redes. En [79] se presenta una metodología UML que, basada en diagramas de secuencia, permite crear modelos para definir ataques a la seguridad de la red.

Sin embargo, hay una carencia en las posibilidades de poder explorar múltiples alternativas de diseño de la red, de una forma rápida y sencilla, lo que permita optimizar la plataforma en condiciones de funcionamiento críticas y normales. Por ejemplo, en [80], se presenta una metodología para modelar y simular el efecto del entorno en las comunicaciones entre sistemas embebidos en una red. Este trabajo permite la verificación de aplicaciones distribuidas en redes. Sin embargo, en este trabajo, la arquitectura interna HW/SW de los nodos no se considera en la simulación.

Esta carencia está cubierta por los autores en [81]. En este trabajo, el modelado previo se complementa con la consideración de la plataforma HW/SW de los nodos. Aquí, el simulador propuesto en [80] se conecta con un co-simulador HW/SW.

Sin embargo, este entorno no considera problemas de seguridad en la red: en todo el flujo de diseño no se consideran los potenciales ataques a la red y, por lo tanto, los efectos sobre el comportamiento de la red no son analizados.

Las redes se ven afectados por su entorno, donde potenciales atacantes provocan vulnerabilidades, generando fallos de seguridad que provocan una reducción en el rendimiento. Por lo tanto, un flujo de diseño de redes debe considerar este entorno activo para poder garantizar el rendimiento de la red.

Para hacer frente a estos requisitos, todas las alternativas de diseño consideradas en el proceso de especificación de la red deberían poder ser simuladas. En este contexto, la simulación nativa proporciona estimaciones de rendimiento rápidas de una configuración específica del sistema. De esta forma, las alternativas de diseño consideradas para la especificación del sistema pueden ser simuladas fácil y rápidamente, obteniendo conclusiones sobre lo correcto o débil de la configuración del sistema seleccionada.

El trabajo desarrollado por J. Jürjens en [82], [83], [84] y [85] permite el análisis de la seguridad en sistemas distribuidos en base a propiedades temporales. En [82] se presenta UMLsec, que es una extensión de UML que permite la creación de modelos para capturar aspectos de la seguridad de la redes, definiendo nuevos estereotipos para modelar aspectos de la seguridad (confidencialidad, enlaces seguros…). Esto permite la evaluación de los modelos UML para detectar debilidades en el diseño de la red usando semántica formal. En [83], se presentan las herramientas que posibilitan la obtención automática de los modelos de evaluación UMLsec, para analizar el sistema y verificar que los requisitos de seguridad se cumplen [85]. Este entorno se aplica al análisis de la seguridad de las comunicaciones móviles [84].

Estos trabajos están orientados al software, proporcionando un entorno para el desarrollo de aplicaciones críticas en lo que a su seguridad se refiere. Mediante diagramas de comportamiento e interacción de UML (actividad, secuencia o estados), el diseñador puede modelar diferentes requisitos de seguridad en escenarios específicos de aplicación, definiendo la secuencia de tareas a ejecutar en este escenario. Luego, el escenario se analiza utilizando las herramientas del entorno de diseño. El trabajo presentado en esta tesis se centra en aspectos estructurales, donde la aplicación y la plataforma HW/SW de los nodos se incluyen en el análisis de la red. Este análisis se centra en el estudio de los efectos causados por los ataques en la red en términos de magnitudes físicas como el consumo de energía o la modificación de la interrupción de las comunicaciones entre los nodos. El trabajo de J. Jürjens se centra en analizar la preservación de la integridad de la transmisión de datos.

De la misma manera, en [86] se presenta un entorno para el análisis de rendimiento de la red. Este entorno está orientado al análisis del software, permitiendo modelar los aspectos de comportamiento del sistema, considerando los recursos de la plataforma. El entorno incluye un conjunto de herramientas para realizar el análisis de rendimiento del sistema capturado en el modelo. Este análisis de rendimiento permite estudiar el impacto de los mecanismos de seguridad introducidos en el sistema para protegerlo de los riesgos introducidos desde el exterior. Los resultados de rendimiento obtenidos incluyen tiempos de ejecución y de servicio (incluyendo retrasos de colas) para recursos de software, y utilización de recursos de hardware y software de la plataforma.

Sin embargo, los objetivos de este análisis de rendimiento son diferentes de los del trabajo presentado en esta tesis. El trabajo de [86] se centra en la simulación del sistema en base a tiempos asociados a las tareas software del programa y al tamaño de los paquetes transmitidos, estando predefinidos y anotados en el modelo UML. En el caso del trabajo presentado en esta tesis, el código real se ejecuta y los tiempos de ejecución y los tamaños de paquetes se obtienen dinámicamente durante la simulación. Por lo tanto, también es posible estimar los efectos que los ataques producen dentro de las funciones del código de aplicación, lo que no es factible con el alto nivel de abstracción que se usa en [86].

3.6 UML/MARTE para múltiples etapas de diseño

Por último, la utilización de UML/MARTE como plataforma común sobre la que realizar más de una etapa del proceso de diseño, que es el núcleo fundamental de esta tesis, también se ha explorado en algunos trabajos. Entre estos trabajo destacan los trabajos desarrollados entorno a Gaspard2 [37][38][39][40]. Gaspard2 es un entorno de

diseño para aplicaciones intensivas en datos, que basándose en un modelo de MARTE, permite describir sistemas MPSoC. Gaspard2 utiliza diagramas compuestos y el perfil MARTE para capturar tanto la estructura de la aplicación como de la plataforma.

La herramienta Gaspard2 soporta el encadenamiento de diferentes herramientas de transformación de modelo a modelo (M2M). Este hecho facilita la realización de síntesis, así como modelos de rendimiento. Además de esto, Gaspard2 soporta la generación de modelos SystemC/TLM, permitiendo realizar simulaciones rápidas, lo que acelera la exploración del diseño. Por ejemplo, en [40] MARTE se aplica de forma más específica para el modelado de hardware. Otro ejemplo puede encontrarse en [41], donde modelos basados en MARTE se usan para describir aplicaciones empotradas de tiempo real.

Además, en [39] se presenta una semántica de control genérica para especificar la adaptabilidad en los sistemas empotrados, especialmente orientado a la reconfigurabilidad en SoCs. La reconfigurabilidad dinámica se implementa generando el código para una región dinámicamente reconfigurable que se relaciona con un modelo de aplicación de alto nivel. Luego, se traduce en funcionalidad HW, generando el código fuente relacionado a un controlador de reconfiguración, que gestiona las diferentes implementaciones asociados con el recurso HW.

Otro enfoque centrado en los múltiples usos de MARTE se puede encontrar en [35]. Aquí se presenta una metodología llamada MopCoM. Esta metodología permite el co-diseño HW/SW de sistemas embebidos y de tiempo real de alta calidad. En este trabajo, se implementa una generación de VHDL. Sin embargo, este hecho no implica que el VHDL sea el único lenguaje que se pueda generar; otros como por ejemplo SystemC también son considerados. Además, la metodología MopCoM puede especificar la semántica de los diferentes modelos de computación por medio de una serie de estereotipos creados ad hoc, que capturen la heterogeneidad semántica de los sistemas [42].

4. Otros Lenguajes y Herramientas Usados en la Tesis

A lo largo de esta tesis se van utilizar un conjunto de herramientas y de lenguajes para su desarrollo. Ahora se van a enumerar y describir. El papel que jugarán cada una de ellas en el trabajo presentado en esta tesis se describirá en más detalle en los capítulos *Metodología de Modelo Único y Entorno de Desarrollo: Integración* .

4.1 Herramientas Utilizadas

Las herramientas que se han integrado en el entorno de desarrollo propuesto en la tesis son:

- Eclipse
- Papyrus
- Acceleo
- SCoPE y VIPPE
- MAST
- MOST
- eSSYN
- Xilinx

Eclipse

Eclipse es una plataforma de software compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado "Java Development Toolkit" (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse). También cubre otro tipo de entornos de aplicación como el "Eclipse Modeling Project", cubriendo casi todas las áreas de "Model Driven Engineering" (MDE).

En el contexto de este trabajo, Eclipse jugará el papel de entrono de desarrollo, en concreto, posee todo lo necesario para poder implementar el conjunto de generadores de código, base en el trabajo de esta tesis, usados para la transformación del modelo UM/MARTE. Además de esto, Eclipse juega el papel de entorno de integración, donde las diferentes herramientas son llamadas a través de un conjunto de "plugins" instalados en él.

Papyrus

Papyrus [13] es una herramienta "Open Source" para UML, basada en Eclipse. Ha sido desarrollado por el Laboratorio de Ingeniería Acelerada por Modelos para

Sistemas Embebidos (LISE), que forma parte de la Comisión de Energías Alternativas y Energía Atómica (CEA-List) de Francia.

Papyrus puede utilizarse como una herramienta independiente o como un complemento de Eclipse. Proporciona soporte para lenguajes específicos de dominio y SysML. Papyrus está diseñado para ser fácilmente extensible, ya que se basa en el principio de perfiles UML. Permite la creación de nuevos mecanismos expresivos para poder capturar aquello que MARTE no puede especificar.

Acceleo

Acceleo [14] es una implementación práctica del lenguaje estándar de la OMG "Model to Text Language" (MTL). Acceleo es el resultado varios proyectos de desarrollo I+D iniciados en la empresa francesa Obeo. La unión entre el estándar OMG MTL, el entorno de desarrollo y los últimos avances de investigación en el campo M2T, ofrece muchas ventajas: alta capacidad de personalización, interoperabilidad, fácil desarrollo de generadores, integración con Eclipse, implementación de plugins…

SCoPE&VIPPE

SCoPE es una herramienta desarrollada en el Grupo de Micro-Electrónica de la Universidad de Cantabria, cuyo desarrollador principal es H. Posadas. SCoPE [19] proporciona una tecnología capaz de realizar la co-simulación de HW/SW temporal, con tiempos de simulación muy bajos. SCoPE combina la creación de modelos de plataforma para permitir la ejecución de SW sobre ellos. También proporciona una técnica de simulación alternativa a los simuladores de conjuntos de instrucciones (ISS), obteniendo velocidades de simulación de factores sobre x100 a cambio de errores de estimación de rendimiento de aproximadamente 10%.

SCoPE es una herramienta especialmente orientada a los primeros pasos del desarrollo de sistemas empotrados:

- Explorar las mejores configuraciones del sistema. SCoPE permite el modelado rápido de sus componentes SW y HW en un modelo de plataforma completo. Por lo tanto, puede analizar los resultados de rendimiento de sus posibles configuraciones, explorando los efectos de diferentes componentes, como procesadores, y de las propiedades de estos (como frecuencias o tamaños de cache, etc.).

- Poner a disposición de los diseñadores SW una plataforma virtual donde se puede considerar la interacción con componentes HW. Por lo tanto, el desarrollo del SW puede comenzar sin requerir un prototipo de la plataforma HW.

SCoPE también es el simulador usado para analizar las redes de sensores y los efectos de atacantes externos sobre ella. En [87] se presenta la extensión de este simulador para permitir realizar dicho análisis. Está extensión se SCoPE se desarrolló en el Grupo de Micro-Electrónica de la Universidad de Cantabria, cuyo desarrollador principal es A. Díaz.

SCoPE fue modificado, evolucionando hasta una nueva versión, que se llamó VIPPE [21]. VIPPE es una herramienta que se desarrolló en el Grupo de Micro-Electrónica de la Universidad de Cantabria, cuyo desarrollador principal es L. Díaz. VIPPE se basa en la tecnología llamada simulación nativa paralelizada que permite a VIPPE explotar el paralelismo subyacente de las plataformas de desarrollo actuales.

SCoPE y su versión actualizada VIPPE son los simuladores que se usarán para el análisis de prestaciones, exploración del espacio de diseño y como herramienta de análisis de redes de sensores.

MAST

MAST es [24] es un conjunto de herramientas de código abierto para realizar análisis de planificabilidad de sistemas distribuidos de tiempo real, evaluando una gran variedad de requisitos temporales. A través del análisis de sensibilidad, se podrá saber hasta dónde o como de cerca está el sistema de cumplir con esos requisitos temporales. MAST fue desarrollada por el Grupo de Ingeniería Software y Tiempo Real de la Universidad de Cantabria.

MAST es la herramienta usada para análisis de planificabilidad de nuestro sistema.

MOST

MOST [27] es la herramienta desarrollada por el "Dipartimento di Elettronica e Informazione" del Politécnico de Milán. MOST es usada específicamente para permitir la exploración del espacio de diseño de arquitecturas HW/SW y obtener una configuración optimizada de nuestro sistema.

MOST es una herramienta de exploración del espacio de diseño que ayuda al diseñador en la búsqueda de soluciones casi óptimas al problema de la exploración arquitectónica de forma automática. El resultado proporcionado por MOST es un conjunto de configuraciones de Pareto dentro del espacio de evaluación de diseño de la arquitectura dada, y un análisis sobre los efectos de las variables del espacio de diseño consideradas sobre el rendimiento del sistema.

MOST es la herramienta desarrolla para, junto con SCoPE y VIPPE, automatizar la exploración del espacio de diseño.

eSSYN

eSSYN es una herramienta que se desarrolló en el Grupo de Micro-Electrónica de la Universidad de Cantabria, cuyo desarrollador principal es A. Nicolás, con la supervisión de H. Posadas. eSSYN [26] es la herramienta que se utiliza para hacer la síntesis de SW a partir de la información capturada en el modelo y del código funcional asociado. A partir de la información captura en el modelo, esta herramienta es capaz de implementar toda la estructura concurrente, así como todos los mecanismos de comunicación necesarios. Para poder realizar esto, eSSYN es capaz de utilizar primitivas proporcionadas por varias APIs, consideradas en este entorno de desarrollo.

Xilinx

Xilinx Inc. [29] es una compañía de tecnología americana, centrada principalmente en la distribución de dispositivos lógicos programables. Es reconocida por inventar las FPGAs (Field Programmable Gate Array) y también por ser la primera compañía con modelos de manufactura “fabless”. Fundada en Silicon Valley en 1984, su sede está ubicada en San José, California, con oficinas adicionales en diversos continentes. Posee una de las mayores familias de productos de FPGAs, incluyendo las series Virtex (alto rendimiento), Kintex (rango medio), y Artix (bajo costo), y la retirada Spartan (bajo costo).

También posee software computacional Xilinx ISE y Vivado Design Suite. Como consecuencia, se utilizarán las funcionalidades proporcionadas por XILIN como conjunto de herramientas para la síntesis de HW.

4.2 Lenguajes

Los lenguajes y paradigmas utilizados durante el flujo de diseño son, además de UML/MARTE:

- C/C++
- VHDL
- SystemC
- HetSC
- ForSyDe

C/C++

C/C++ son los lenguajes de programación usados en la implementación del código funcional usado para la síntesis de SW.

C es un lenguaje de programación originalmente desarrollado por Dennis Ritchie entre 1969 y 1972 en los Laboratorios Bell, como evolución del anterior lenguaje B, a su vez basado en BCPL. Se trata de un lenguaje de tipos de datos estáticos, débilmente tipificado, de medio nivel, ya que dispone de las estructuras típicas de los lenguajes de alto nivel pero, a su vez, dispone de construcciones del lenguaje que permiten un control a muy bajo nivel. Los compiladores suelen ofrecer extensiones al lenguaje que posibilitan mezclar código en ensamblador con código C o acceder directamente a memoria o dispositivos periféricos. Como resultando es muy utilizado en sistemas empotrados.

C++ es un lenguaje de programación diseñado a mediados de los años 1980 por Bjarne Stroustrup. La intención de su creación fue el extender al lenguaje de programación C con mecanismos que permiten la manipulación de objetos. Posteriormente se añadieron facilidades de programación genérica, que se sumaron a los paradigmas de programación estructurada y programación orientada a objetos. Por ello es habitual en entornos basados en componentes, como el considerado en esta tesis.

VHDL

VHDL es un lenguaje definido por el IEEE ("Institute of Electrical and Electronics Engineers") usado para describir circuitos digitales. VHDL es el acrónimo que representa la combinación de VHSIC y HDL, donde VHSIC es el acrónimo de "Very High Speed Integrated Circuit" y HDL es a su vez el acrónimo de "Hardware Description

Language". Para el modelado físico existe la derivación del lenguaje VHDL-AMS. Originalmente, el lenguaje VHDL fue desarrollado por el departamento de defensa de los Estados Unidos a inicios de los años 80's basado en ADA, con el fin de realizar simulación de circuitos eléctricos digitales; sin embargo, posteriormente se desarrollaron las herramientas de síntesis e implementación en hardware a partir de los archivos .VHD. Aunque puede ser usado de forma general para describir cualquier circuito digital se usa principalmente para programar PLD ("Programable Logic Device" - Dispositivo Lógico Programable), FPGA ("Field Programmable Gate Array"), ASIC y similares.

En el contexto de esta tesis, VHDL es el lenguaje usado para realizar la síntesis de HW.

SystemC

SystemC [28] es un lenguaje que se creó añadiendo a C++ los elementos necesarios para el modelado y diseño de sistemas electrónicos. Como resultado, SystemC es el lenguaje de programación más extendido para ESL, ya que permite la especificación a un alto nivel de abstracción del sistema, su exploración arquitectural, modelado de escenarios para estudiar su de rendimiento, desarrollo de software, verificación funcional y síntesis de alto nivel.

En el contexto del trabajo de esta tesis, SystemC va a ser usado como lenguaje con el cual se van a generar especificaciones ejecutables, utilizadas para realizar simulaciones funcionales del sistema.

HetSC

HetSC ([10], [88]) es una metodología de especificación para sistemas embebidos, heterogéneos y concurrentes en SystemC desarrollada por F. Herrera dentro del Grupo de Micro-Electrónica de la Universidad de Cantabria. HetSC hace una clara separación entre la computación y los aspectos de comunicación del sistema, permitiendo la creación de especificaciones ejecutables formales del sistema. Además, HetSC proporciona los mecanismos de comunicación para capturar la semántica de Modelos de Computación (MoCs) específicos.

En esta tesis se va a utilizar HetSC como metodología que proporciona implementaciones de canales en SystemC que capturan la semántica ejecutiva de un conjunto de MoCs.

Esta metodología presenta dos niveles básicos. En la Figura 3, el primer nivel representa reglas y guías de uso, llamada Metodología General de Especificación. Este nivel facilita y hace más clara y segura la especificación de sistemas concurrentes. Estas reglas facilitan la aplicación actividades en el diseño a nivel de sistema, como por ejemplo análisis de funcionamiento a nivel de sistema. Esto facilita la aplicación de un flujo de co-diseño, como generación de software, síntesis de hardware y generación de interfaces HW/SW. La Metodología General de Especificación también define una representación gráfica para constructores de SystemC que ayuda al usuario crear de manera inequívoca una especificación en SystemC.

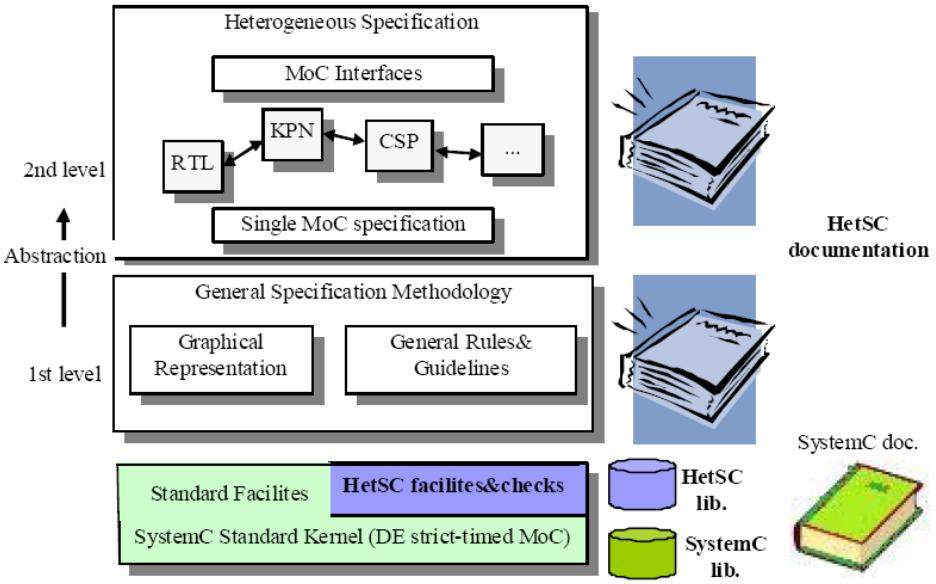


Figura 3 La metodología HetSC vista como dos niveles.

Estos conceptos son manejados en el segundo nivel. Este nivel está especificado en "Heterogeneous Specification" de la Figura 3. En la primera parte, se relata como el constructor modela bajo un dominio específico de diseño, lo que es llamado "single-MoC specification" en la Figura 3. HetSC contempla diferentes posibilidades; el usuario puedo querer construir:

- Modelos abstractos y concurrentes especifican redes de procesos funcionales, comunicados por medio de elementos de comunicación muy abstractos, como FIFOs, "rendezvous", etc. Aplicaciones básicas son modelos de redes, software embebido y concurrente, etc. Para cubrir estos ámbitos, la metodología HetSC proporciona los MoCs atemporales.

- Modelos reactivos, donde el sistema reacciona; computa como respuesta a estímulos procedentes del ambiente tan rápido como le sea posible, tomando tiempo cero en ello en la aproximación ideal. Esto es típico para software reactivo. Para cubrir este ámbito, la metodología HetSC proporciona el MoC síncrono reactivos, que es un tipo de MoC síncrono.
- Modelos síncronos de reloj, donde las computaciones toman un cierto lapso de tiempo dado en eventos de reloj. Como ejemplo de esto, los modelos de hardware digital. Para cubrir este ámbito, la metodología HetSC proporciona el MoC síncrono de reloj, que es otro MoC síncrono.
- Modelos análogos, compuesto de componentes analógicos, como redes de componentes de circuitos como resistores, transistores, etc. Para cubrir este ámbito, la metodología HetSC permite su conexión con otras metodologías basadas en SystemC, como SystemC-AMS

La librería HetSC proporciona un conjunto de mecanismos para cubrir las deficiencias del lenguaje SystemC para la especificación heterogéneas. Para soportar varios MoCs específicos, se necesitan de un conjunto de nuevos mecanismos que no están incluidos en SystemC. Estos mecanismos tienen el contenido semántico concreto y el nivel de abstracción requerido para el correspondiente MoC. Además de esto, la librería de HetSC proporciona mecanismos para detectar y localizar violaciones en las reglas de implementación de cada MoC.

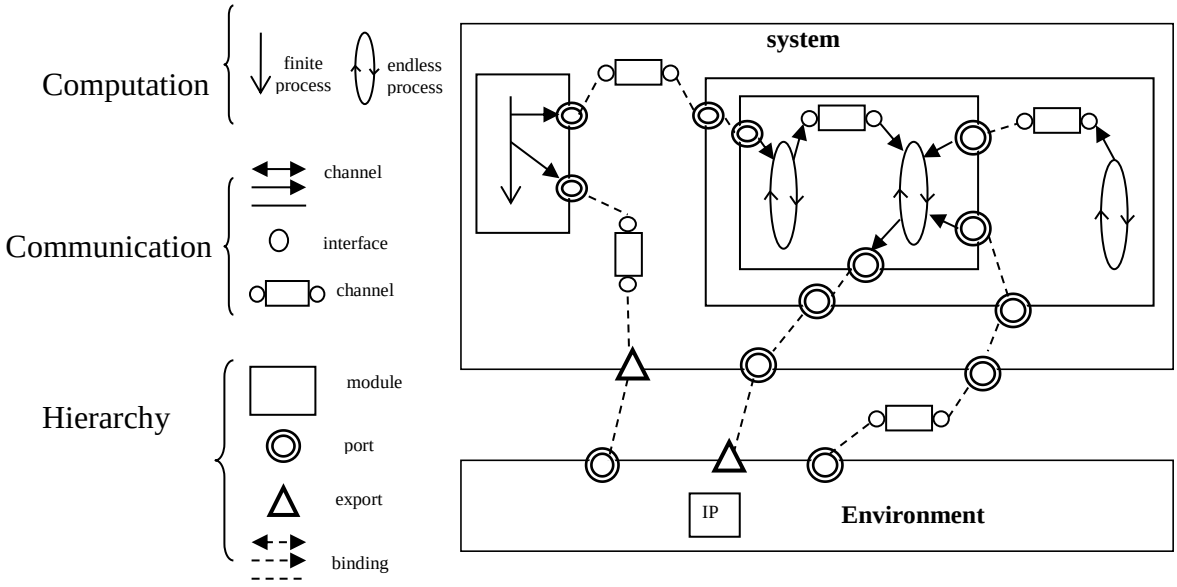


Figura 4 Representación visual de un sistema con HetSC identificado el tipo de cada elemento.

HetSC hace una distinción entre los diferentes elementos que componen el lenguaje SystemC.

El primer conjunto de elementos son los encargados de proporcionar la estructura y la jerarquía al sistema. Estos identifican las diferentes partes que conforman el sistema, singularizándolas según criterio del diseñador.

El segundo de estos conjuntos engloba a los elementos que realizan la transmisión de la información. En este conjunto, diferentes canales serán desarrollados para poder plasmar la específica semántica de un conjunto de MoCs.

Finalmente, el último conjunto agrupa a los elementos dónde la computación está localizada. En este caso son los "sc_thread". HetSC hace la distinción entre procesos infinitos, aquellos que están siempre activos, y los procesos finitos, con un único ciclo de ejecución.

ForSyDe

ForSyDe [9] es un metamodelo desarrollado por A. Jantsch e I. Sander y que permite una descripción formal de un sistema. ForSyDe se centra principalmente en la comprensión de la concurrencia y del tiempo, representando el sistema como un conjunto de procesos concurrentes comunicándose a través de señales.

Los procesos y las señales son conceptos del metamodelo con una definición matemática precisa y sin ambigüedades. Una señal ForSyDe es una secuencia de eventos en los que cada evento tiene una etiqueta y un valor. La etiqueta denota la posición del evento en la señal y se usa para denotar orden parcial entre eventos. Los procesos representan relaciones matemáticas entre señales. Los procesos son elementos concurrentes con una máquina de estado interna. La relación entre los procesos y las señales se muestra en la Figura 5.

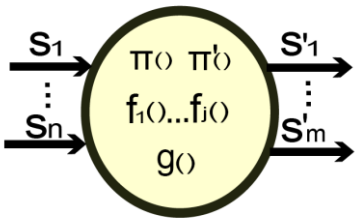


Figura 5 Representación del metamodelo ForSyDe

Un proceso ForSyDe *p* está definido por la expresión:

$$p(s_1 \ldots s_n) = s'_1 \ldots s'_m$$

El proceso *p* toma como entradas un conjunto de señales $(s_1 \ldots s_n)$ produciendo un conjunto de señales de salida $(s'_1 \ldots s'_m)$, donde:

$$\forall\, 1 \le i \le n \ \wedge 1 \le j \le m\, n, m \in \mathbb{N}_0$$

$$s_i, s'_j \in S$$

Siendo s señales individuales y S el conjunto de todas las señales ForSyDe, que pueden ser atemporales, síncronas y temporales.

Las señales de salida son determinadas por las funciones $f_1() \ldots f_j()$ que dependen de las señales de entrada y del estado interno del proceso. El estado interno del proceso está definido por la función siguiente-estado *g()* que depende de las entradas y del actual estado interno del proceso, ω_j.

Capítulo III
Metodología de Modelo Único

Como se ha descrito en la *Introducción*, durante el proceso de diseño se ha de manejar mucha información sobre el sistema con objeto de obtener un resultado final capaz de garantizar su funcionamiento, maximizando su rendimiento. Esto da lugar a la necesidad de considerar una gran cantidad de detalles durante dicho proceso, tanto en etapas de diseño como de análisis. El problema es que las herramientas de diseño no están orientadas a cubrir en profundidad todos y cada uno de los detalles. En consecuencia, el diseñador necesita utilizar diferentes herramientas para poder realizar cada una de las actividades que componen su flujo de diseño: desde el propio modelado del sistema, pasando por todo lo necesario para el análisis de dicho diseño, así como toda la batería de recursos requeridos para su síntesis e implementación. Además, el desarrollo de todas estas actividades implica el manejo de una gran cantidad de información.

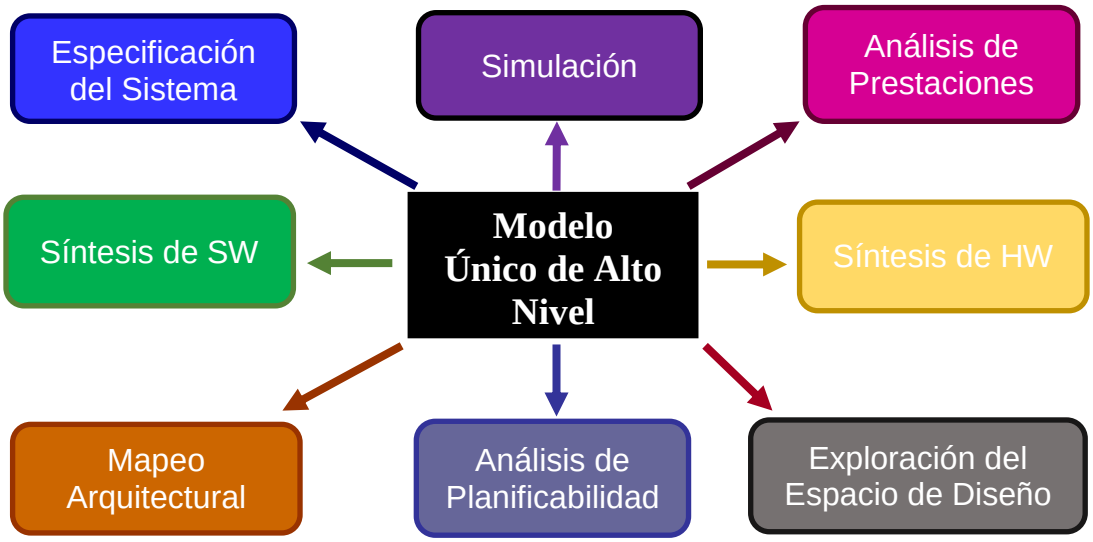


Figura 6 Modelo Único

Como consecuencia, uno de los mayores retos que se presentan durante el proceso de diseño es cómo capturar toda esa información necesaria para realizar el diseño del sistema empotrado, reutilizándola, adaptándola y actualizándola conforme avanza el flujo de diseño, abordando el propio diseño del sistema así cómo los diferentes escenarios para su análisis. De esta forma, es necesario conseguir que las herramientas estén lo más interrelacionadas e integradas posible, con el objetivo de no tener que hacer esfuerzos extras a la hora de migrar de una a la otra, por ejemplo, adaptando formatos de archivos auxiliares para el uso de cada herramienta. También es un gran problema en sí mismo la utilización de múltiples herramientas, ya que es necesario evitar que se requiera de

grandes esfuerzos en lo que la interacción con el usuario se refiere y que este pueda realizar una u otra tarea de la manera más fácil y manejable.

Como solución a todo esto, el trabajo que se presenta en esta tesis quiere demostrar cómo a partir de un único modelo de alto nivel (concepto e idea aportada por E. Villar) se puede establecer un entorno de diseño que permita el desarrollo de un sistema empotrado, abarcando el conjunto de tareas que deban realizarse durante el desarrollo del mismo (Figura 6, figura realizada en colaboración con E. Villar).

El uso de este modelo único permite la utilización, de forma fácil y manejable, del conjunto de herramientas que se requieren según la etapa en la que se encuentre el proceso de diseño y el objetivo del diseñador. Por ello, este modelo único tiene como objetivo capturar toda la información relevante necesaria para caracterizar complemente el sistema, incluyendo la aplicación, la plataforma HW/SW y el mapeo arquitectural, así como poder establecer diferentes tipos de análisis del sistema que permitan su correcto desarrollo. Mediante este proceso de evaluación y análisis se puede ir determinando y optimizando la configuración final del sistema. Para ello se utilizan tanto análisis estáticos, como modelos virtuales o mediante la realización de prototipos reales del mismo, mediante un proceso de síntesis que, según sea el caso, abarque el SW, el HW o ambos.

Para poder realizar todas estas tareas, desde este modelo único se establece un enlace automático con un conjunto de herramientas que permiten realizar dichas actividades de diseño (idea aportada por E. Villar). De esta forma, el diseñador tiene un entorno de desarrollo completo que permite la especificación de su sistema, diseñándolo, analizándolo e implementándolo paso a paso, de forma fácil, ágil e integrada.

Para conseguir este propósito, la tesis propone la realización de este modelo de alto nivel mediante una metodología basada en UML/MARTE, que se describe a continuación brevemente y se detalla en el capítulo *Modelado de Sistemas Electrónicos*.

Además, durante la tesis se ha desarrollado una infraestructura que demuestra estos conceptos en la práctica. Para ello se ha trabajado con diversas herramientas que cubren distintas etapas del proceso de diseño y distintos tipos de análisis, desde análisis estáticos a generación de prototipos. Dichas herramientas se han integrado en un entorno global, de forma que la información del modelo es transferida automáticamente a las herramientas para que puedan ser ejecutadas de la manera más rápida posible, sin que el ingeniero tenga que perder tiempo transfiriendo y adaptando la información de unas herramientas a otras.

La forma en la que este modelo se enlaza con las herramientas se mostrará en el capítulo *Entorno de Desarrollo: Integración* .

1. Etapas de diseño

Para realizar una metodología de modelo único lo suficientemente amplia se ha considerado distintas etapas habituales en el diseño de sistemas empotrados. Así, las etapas del proceso de diseño de un sistema empotrado en el que se sustenta el entorno de desarrollo presentado en esta tesis se puede estructurar en los bloques que se muestran en la Figura 7 (figura hecha en colaboración con H. Posadas).

Siguiendo esta idea, primero se crea el modelo del sistema a diseñar, incluyendo aquellos aspectos del mismo que se quiere analizar e implementar en función de los requisitos funcionales y no funcionales a implementar por nuestro sistema. Una vez realizado el modelo de alto nivel, se deben de realizar la batería de test que se consideran para comprobar que el diseño realizado cumple con los requisitos del sistema.

Una vez el sistema ha sido modelado completamente, y se ha definido los entornos de prueba, es momento de comenzar con el análisis y refinado del sistema. Para ello, los tipos de análisis cuberitos se pueden agrupar en tres tipos, como se ve en la Figura 7:

- El análisis funcional se utiliza para validar la correcta implementación de la funcionalidad de nuestro sistema a nivel de aplicación.
- En el análisis con modelo virtual se estudian las prestaciones del sistema; se toma la aplicación ya verificada funcionalmente y se le asocia un modelo de la plataforma para estudiar las prestaciones del sistema. Aquí se exploran diferentes configuraciones de sistema como número de recursos de cómputo o el mapeo arquitectural.
- Una vez hecho este análisis, se hace un prototipo del sistema mediante un proceso de síntesis, con el cual se vuelve a validar el diseño del sistema sobre la plataforma real destino.

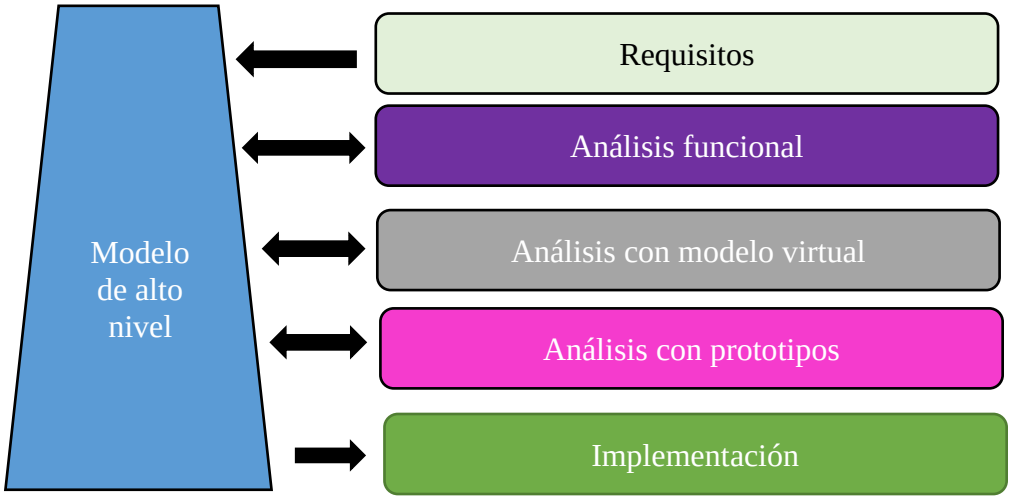


Figura 7 Áreas del proceso de desarrollo

Finalmente, después de realizar los análisis y procesos de refinado que se consideren oportuno, se puede obtener la implementación final del sistema.

En el contexto de esta tesis, estas fases se han implementado en base a diferentes casos concretos, los cuales se muestran con más detalle en la Figura 8. En dicha figura se muestra la estructura del entorno de diseño que se propone en esta tesis. Así pues, la Figura 8 muestra la estructura y las herramientas involucradas en las diferentes tareas mostradas en la Figura 7. En esta Figura 8, los recuadros rojos que hay en cada sub-área de trabajo son las distintas herramientas y lenguajes que se usan para el desarrollo de dicha tarea. Todos estos lenguajes/herramientas se presentaron en la sección *Otros Lenguajes y Herramientas Usados en la Tesis* del capítulo *Estado del Arte*.

1.1 Modelado

En el modelo se capturan la aplicación y la plataforma, realizando el posterior mapeo arquitectural, como se representó en la Figura 1. Toda la metodología de modelado se describirá en el capítulo *Modelado de Sistemas Electrónicos*.

La aplicación es construida en base a los principios formales de ForSyDe (sección *Análisis: Simulación funcional* del capítulo *Entorno de Desarrollo: Integración de Herramientas*), que también se usará para establecer un tipo de análisis estático, así como para dar soporte a la generación de especificaciones ejecutables para el análisis (sección *Análisis: Simulación* del capítulo *Entorno de Desarrollo: Integración de Herramientas*).

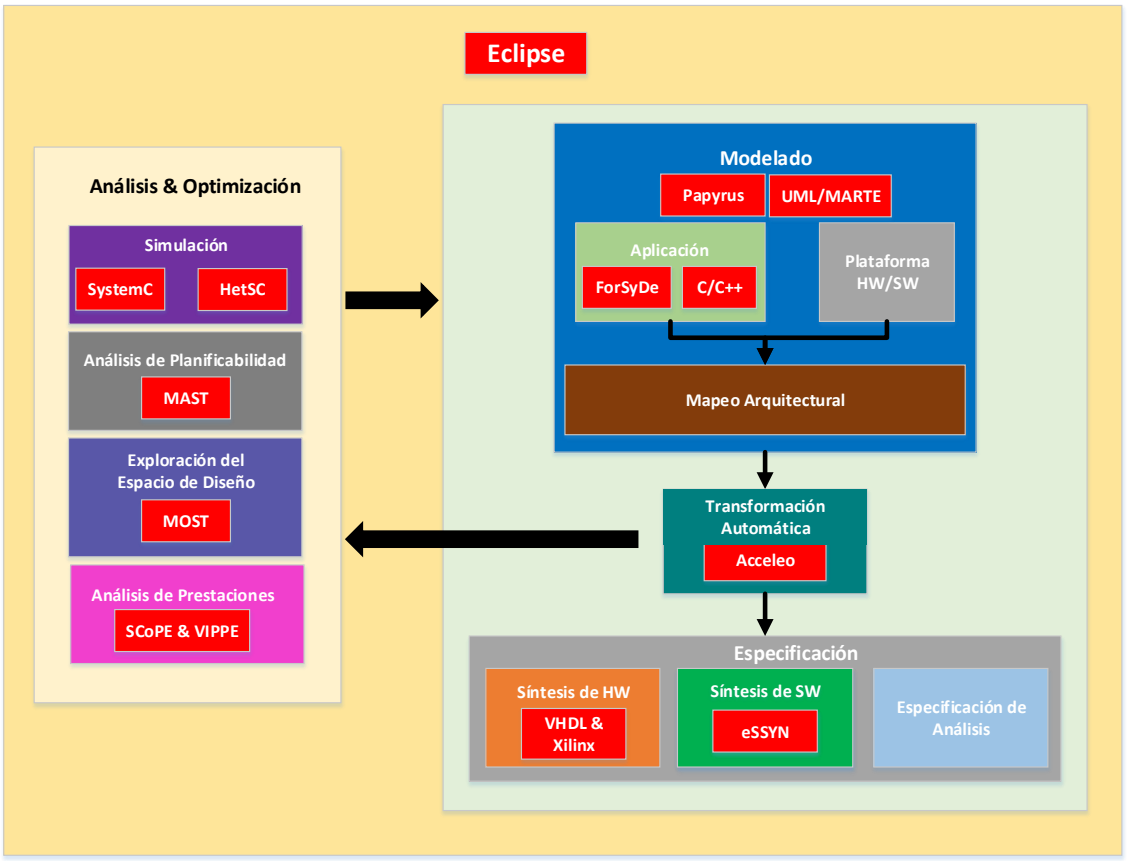


Figura 8 Esquema del entorno de diseño

Como se explicará más adelante, a la aplicación se le puede asociar el conjunto de archivos de código que implementa su funcionalidad o diagramas de los que pueda sintetizarse el código funcional. Los lenguajes seleccionados para implementar este código son C y C++.

La plataforma es capturada en base a elementos HW y SW bien caracterizados con sus propiedades más relevantes. Una vez descrita la plataforma, se realiza el mapeo arquitectural de la aplicación sobre los recursos de cómputo presentes en dicha plataforma.

Respecto al modelado, la herramienta utilizada es Papyrus. Papyrus es la herramienta que se ha utilizado para capturar el modelo UML/MARTE en este trabajo. Además de esto, se ha usado para implementar nuevos mecanismos expresivos; por ejemplo los canales (sección *Canales y Concurrencia* del capítulo *Modelado de Sistemas Electrónicos*) o los atacantes que deben soportar las redes de sensores (sección *Ataques a*

Redes del capítulo *Modelado de Sistemas Electrónicos*), así como realizar librerías de componentes para facilitar el desarrollo de los modelos.

Modelado de la funcionalidad

La funcionalidad se organiza en componentes interconectados mediante mecanismos de comunicación definidos por un conjunto de propiedades que determinan cómo se va a transmitir la información. La funcionalidad de cada componente, proporcionada mediante archivos fuente o diagramas, debe ser independiente de plataforma, para poder permitir las posteriores actividades de mapeo, análisis y exploración. Además, esta independencia facilita la reutilización de los componentes entre distintos diseños.

En consecuencia, todas las actividades que requieran de recursos de la plataforma, tales como mecanismos de comunicación o generación de hilos de ejecución, deben ser definidas explícitamente en las descripciones estructurales de la aplicación, y no en los códigos.

Como resultado de estas propiedades añadidas a la estructura de la aplicación, a través de las características de los componentes, sus interfaces o los canales que los interconectan se define la estructura concurrente de la implementación final, identificando que componentes serán activos, asociados a su propio hilo de ejecución, y cuales pasivos.

Modelado de la plataforma HW/SW

Una segunda tarea a cubrir en el proceso de diseño del sistema es la de determinar la plataforma destino donde la aplicación va a ser ejecutada. En este punto, el diseñador puede optar por seleccionar plataformas presentes en el mercado y, mediante un proceso de síntesis de SW, adaptar la aplicación a dichas plataformas y estudiar sus prestaciones. Otra estrategia es realizar simulaciones de la aplicación sobre plataformas, generando modelos virtuales para simulación, las cuales permiten explorar diferentes parámetros de la plataforma como número de procesadores, frecuencias de los mismos, tamaños de cache… Con esta información tomar la decisión sobre la plataforma, pudiendo realizar un nuevo análisis de la misma mediante la utilización, de nuevo, de la síntesis de SW.

Modelado del mapeo arquitectural

El mapeo arquitectural consiste en asociar los componentes de la aplicación a los diferentes recursos de la plataforma definida en el paso anterior. En este punto, los componentes de aplicación pueden ser mapeados a un recurso u otro. También puede estudiarse si un componente se implementa como SW (asociándolo a un sistema operativo) o como HW (asociándolo, por ejemplo, a una FPGA) con el fin de obtener información sobre rendimiento, consumo, dependencias... De esta forma, dependiendo de mapeo, se requerirán diferentes tipos de recurso para la implementación del sistema como compiladores, librerías, herramientas, APIs… que se deberán considerar e incluir en el entorno de diseño.

1.2 Transformación Automática

A partir de la información captura en el modelo, es posible realizar las diferentes tareas que puede necesitar el diseñador, como la síntesis de código funcional o de código de comunicaciones y mapeo o la optimización del sistema mediante la realización de algún tipo de análisis. Para poder hacer esto, la información capturada en el modelo ha de ser transferida a las herramientas correspondientes. Para ello es necesario transformar la información al formato requerido por cada una de las herramientas y que habitualmente se define por sus elementos de entrada. Dicha transformación se hace mediante procesos de generación automática. Estas tareas se llevan a cabo mediante procedimientos que se basan en el uso de la herramienta Acceleo, integrada en Eclipse y que tiene fácil acceso a todos los detalles de los modelos UML/MARTE.

1.3 Generación de versiones ejecutables para el análisis

La transformación automática del modelo para generar modelos analizables puede hacerse de dos formas: generando los archivos de descripción que sirvan de entrada a las herramientas auxiliares encargadas de crear los modelos y los análisis o generando los modelos propiamente dichos.

El primer escenario conlleva que, del resultado de la transformación, se creen unos archivos intermedios con la información del modelo. A partir de estos, las correspondientes herramientas leen tales archivos y realizan su correspondiente tarea. Por ejemplo, este es el escenario que se realiza para la síntesis de SW.

El segundo se usa para que, desde el modelo se obtengan códigos ejecutables directamente. También se puede englobar en este escenario la síntesis de HW, ya que se

generará toda la infraestructura necesaria (en VHDL) para realizar dicha síntesis mediante el entorno que proporciona Xilinx.

Para la tarea de síntesis se ha de utilizar un sintetizador de código que, a partir de la información capturada en la especificación de alto nivel, se genere toda la infraestructura de código que implementa las propiedades capturadas en dicha especificación, permitiendo la compilación y la consecuente obtención de los ejecutables.

Para este fin, en el modelo del sistema se ha de incluir todos aquellos aspectos necesarios para realizar la compilación del SW generado; tales como compiladores, opciones de compilación y enlazado, librerías auxiliares, código funcional… De esta forma, la automatización del proceso se podrá realizar.

Es importante notar que en la mayoría de actividades propuestas en la tesis, ambos elementos deben combinarse. Esto se debe a que varias de las herramientas utilizadas requieren tanto los códigos de las aplicaciones a ejecutar como descripciones que incluyen información de los requisitos, plataforma o mapeo arquitectural. Debido al requerimiento impuesto por la metodología desarrollada de que los códigos asociados a los componentes sean independientes de plataforma, la generación de los códigos completos de la aplicación requiere de combinar estos códigos con otros códigos dependientes de plataforma, donde se utilicen los detalles de la plataforma destino para implementar los elementos necesarios para el despliegue, tales como las comunicaciones o la creación de los flujos de ejecución. Por tanto, la generación de la aplicación SW, ya sea para su ejecución en modelos de alto nivel, plataformas virtuales, o prototipos físicos implica un proceso de síntesis de código. La herramienta que realiza la síntesis de SW es eSSYN.

1.4 Análisis

Con el fin de dotar de una mayor versatilidad, la metodología que se propone en esta tesis se ha cubierto la realización de diversos tipos de análisis. De esta forma el diseñador tiene a su disposición un amplio abanico posibilidades para estudiar su diseño y sacar conclusiones, lo que facilita el proceso de refinamiento de su diseño. Todo depende de qué aspecto quiera estudiar el diseñador. Esto puede depender de la fase del diseño en la que se esté o de los objetivos del proyecto que se pretenda realizar.

Análisis Funcional

Un primer análisis que se ha cubierto en esta tesis es un análisis puramente funcional. En él se considera únicamente la aplicación, independiente de la plataforma destino donde se mapeará dicha aplicación. Este hecho permite una exploración de la estructura funcional, al poder estudiar el comportamiento del sistema atendiendo a su estructura de componentes, las propiedades de los mecanismos de comunicación y la concurrencia asociada, detectando problemas como bloqueos, inanición u otros problemas derivados de una mala implementación de la aplicación.

La especificación ejecutable para el análisis generada desde el modelo se implementa usando el lenguaje SystemC. En un caso específico, esta especificación SystemC está complementada mediante HetSC, como se verá en la sección *Análisis: Simulación funcional* del capítulo *Entorno de Desarrollo: Integración de Herramientas*, donde se discutirá la utilización del modelo UML/MARTE para poder analizar diferentes semánticas de comunicación asociadas a MoCs.

Análisis Prestaciones del Sistema

Otro tipo de análisis cubierto en esta tesis es el análisis de prestaciones, donde se tiene la necesidad de considerar la plataforma. Éste análisis permite estudiar el funcionamiento del sistema en términos no funcionales, tales como consumo, latencia, ancho de banda, calidad de servicio... Además, este análisis permite la obtención de valores temporales de la ejecución de las funciones de la aplicación, que son necesarios para hacer otro tipo de análisis, como el de planificabilidad.

Estos análisis permiten la optimización del sistema, pudiendo estudiar el impacto en el rendimiento considerando diferentes mapeos de los componentes de la aplicación, modificando los recursos presentes en la plataforma, propiedades de dichos recursos, como la frecuencia de operación,…

Las tareas de evaluación de cada una de las implementaciones posibles se realizarán mediante la utilización de las herramientas SCoPE y VIPPE.

Además, el conjunto de análisis resultante de considerar las distintas alternativas de diseño posibles define un proceso de exploración del espacio de diseño que tiene como fin último seleccionar un subconjunto de configuraciones del sistema que cumplan los requisitos del diseño. Con este subconjunto, el diseñador decide el mapeo arquitectural

sobre las plataformas destino, considerando los recursos HW/SW disponibles de las mismas.

La tarea de analizar el conjunto de configuraciones de un sistema puede resultar ser una tarea muy engorrosa en función de cuántos aspectos del sistema queramos evaluar. El rendimiento del sistema se evalúa en función de los potenciales valores que pueden tomar las variables de diseño consideradas en la especificación del sistema. Dicha evaluación se puede realizar probando las distintas configuraciones del sistema una por una, o mediante un proceso automático que evalúe cada una de ellas. El resultado de este análisis es la generación de una sería de métricas que muestran si los requisitos del sistema se cumplen en función de las configuraciones del sistema dadas.

Para realizar estas actividades de manera sencilla y automática, la exploración del espacio de diseño se realizará mediante la herramienta MOST.

Análisis de planificabilidad

El análisis de planificabilidad permite evaluar los efectos resultantes de los mapeos seleccionados de componentes funcionales sobre los recursos de la plataforma, incluyendo parámetros como las prioridades de las tareas, para verificar si se cumplen las restricciones temporales que garantizan que el sistema resultante es planificable. Además de esto, se obtienen estimaciones de tiempo disponible de utilización de los recursos de cómputo, lo que es útil para poder añadir nuevas tareas al recurso de procesamiento, o quitar según sea el caso. Esta última propiedad permite también hacer un análisis de sistemas de criticidad-mixta, donde las tareas a ejecutar presentan prioridades de ejecución diferentes. De esta forma, se puede obtener cuánto utilizan las tareas de más alta prioridad los recursos disponibles de la plataforma, extrayendo cuánto quedaría disponible para ejecutar tareas de baja criticidad.

La herramienta utilizada en la infraestructura desarrollada en la tesis en este caso será MAST.

2. Relaciones entre las distintas etapas

A pesar de que la metodología que aquí se presenta no tiene como objetivo forzar un flujo de diseño en concreto, sí es cierto que para realizar algunas de las actividades que el entorno de desarrollo proporciona es necesario un cierto ordenamiento de las mismas. Estas relaciones de orden se muestran en la Figura 9.

Conforme a la figura, el proceso de diseño parte de la creación del modelo UML/MARTE que contendrá aquella información necesaria que el diseñador considere para su objetivo, y que por tanto debe ser siempre la primera tarea a realizar.

A partir de aquí, se realizará la generación de código, seleccionando el generador que corresponda a los objetivos del proceso de diseño (simulación funcional, síntesis HW…). De estas, hay algunas de las áreas del entorno de desarrollo que requieren de información de otras y que, por tanto, han de ser realizadas con anterioridad. Esto ocurre específicamente con el análisis de presentaciones y la síntesis SW, ya que el SW es necesario para poder ser ejecutado en el modelo de plataforma y obtener estimaciones de rendimiento.

De la mismas forma, para el análisis de planificabilidad hace falta datos sobre los tiempos de ejecución de las funciones que van ejecutar la tareas a analizar. Estos tiempos son proporcionados por la herramienta de análisis de prestaciones.

Finalmente, la exploración del espacio de diseño requiere de la herramienta de análisis de prestaciones para ir analizando cada una de las variables de diseño consideras en la exploración, estudiando cual combinación de ellas es la que presenta un mejor resultado respecto al conjunto de métricas seleccionadas para ser optimizadas.

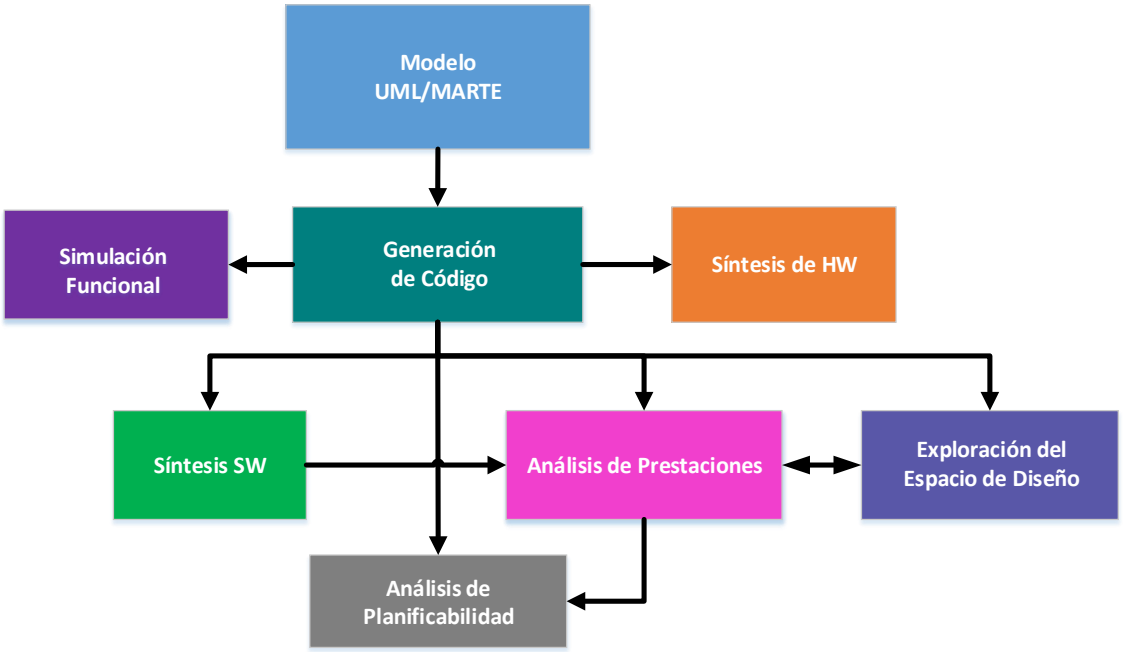


Figura 9 Relaciones las áreas del entorno de desarrollo

Capítulo IV

Modelado de Sistemas Electrónicos

El modelado integral con UML/MARTE de sistemas empotrados presenta una gran cantidad de retos a los cuales hay que enfrentarse. Como se ha ido mostrando en los capítulos anteriores, no solo hay que capturar la aplicación, la plataforma HW/SW y el correspondiente mapeo, también hay que añadir toda la información complementaria que permita la realización de los diferentes tipos de análisis considerados en este entorno de desarrollo.

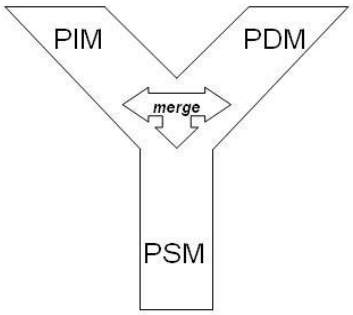


Figura 10 Flujo en Y de las metodologías MDD

Sin embargo, gestionar tanta información no es sencillo. Por ello, toda la información que se ha de capturar en el modelo de sistema empotrado ha de estar adecuadamente estructurada para poder abordarla con cierta facilidad. Así se simplificará su uso en las tareas de creación, visualización, análisis y actualización del modelo, aumentando la eficiencia del proceso de diseño.

Una de las soluciones adoptadas para estructurar la información del modelo se basa en considerar en qué etapas del proceso de diseño es utilizada cada pieza de información. En este sentido, UML/MARTE proporciona la suficiente capacidad expresiva para poder establecer el tradicional flujo en 'Y' de las metodologías de diseño basadas en modelos ("Model-Driven Design", MDD) [4], donde el modelo es dividido en tres sub-modelos:

- El Modelo Independiente de Plataforma ("Platform Independent Model", PIM), que describe los aspectos funcionales y no funcionales del comportamiento de la aplicación (por ejemplo, estructura de la aplicación, el modelado funcional).
- El Modelo de Descripción de la Plataforma ("Platform Description Model", PDM), que describe los diferentes recursos de hardware y software que componen la plataforma donde se ejecutará el sistema.
- El Modelo Específico de Plataforma ("Platform Specific Model", PSM), describe la arquitectura de la plataforma. Además de esto, este modelo se captura la

asignación de las funcionalidades del sistema a los recursos de la plataforma en los cuales van a ser ejecutados.

Como se va ir mostrando a lo largo de esta tesis, los aspectos a abordar en el modelado de cada uno de estos sub-modelos del sistema son muy amplios y variados. Por ello, en las diferentes secciones que componen este capítulo, cada uno de estos aspectos se describirá más en detalle, mostrando qué papel juegan en la creación del modelo.

Además, a lo largo de este documento, se ha puesto el énfasis más en qué información debe ser capturada para soportar cada una de las tareas o herramientas que en los elementos específicos de UML o MARTE utilizados, como estereotipos y atributos, aspectos clave de estos lenguajes. Con ello, se pretende no saturar al lector con listas de atributos y estereotipos que pueden hacer perder la perspectiva y el foco de lo que se puede realizar con un modelo UML/MARTE a la hora de diseñar sistemas empotrados, además de evitar que el documento resultante sea excesivamente largo.

Para más información en cada uno de estos detalles, cabe destacar que a lo largo del desarrollo de esta tesis se han ido creando un conjunto de manuales donde se describen y detallan la gran mayoría de los aspectos de la metodología de modelado que aquí se resume (referencias [nn)], [oo)], [pp)], [qq)], [rr)], [ss)], [tt)], [uu)]). En dichos manuales se describen de forma detallada todos los elementos de UML y de MARTE utilizados durante el desarrollo de esta tesis, así como el cómo han de usarse en el contexto de esta metodología de modelado. Estos manuales se han realizado principalmente por mí, con la colaboración de F. Herrera y la supervisión de E. Villar.

A lo largo de este capítulo, además de referencias a los artículos publicados, se hará referencia a estos manuales, lo cual permitirá al lector tener una descripción más detallada de cada uno de los aspectos metodológico (estereotipos y atributos y formas de usarlos y de especificarlos) referidos en las respectivas subsecciones que componen este documento.

De la misma manera, esta tesis ha sido realizada en el contexto de un conjunto de proyectos de investigación como se mostró en la sección *Proyectos de Investigación* del capítulo *Introducción*. En cada sección se incluirá el proyecto o proyectos donde dicho aspecto del entorno de desarrollo se realizó, así como las publicaciones relacionadas con dicha sección.

1. Metodología de Modelado

La metodología de modelado presentada en esta tesis tiene como objetivo permitir la especificación de todos los aspectos relevantes de un sistema empotrado, especialmente aquellos que tienen un impacto directo en el funcionamiento y rendimiento del mismo. Todos estos aspectos del sistema se estructuran según la división PIM, PSM y PDM mostrada en la sección anterior.

Además de esto, esta metodología permite el modelado de sistemas distribuidos, pudiendo especificar nodos y la estructura de la red como la interconexión de estos nodos. Los nodos pueden ser detallados internamente, definiendo su plataforma HW/SW y la aplicación que se ejecuta en ellos. Así, un escenario cubierto por la metodología es permitir el mapeo de la aplicación en los diferentes nodos que componen la red, actuando a modo de recursos de cómputo de un sistema empotrado cada nodo de la red.

1.1 Modelo Independiente de Plataforma

Para modelar la aplicación, la solución propuesta se basa en estructurar dicha aplicación mediante la utilización de componentes de aplicación. Estos componentes de aplicación contienen parte de la funcionalidad e intercambian datos de forma ordenada para construir la funcionalidad completa.

El hecho de que este modelado sea independiente de plataforma obliga a que las definiciones tanto de la funcionalidad como de la comunicación sean también independientes de plataforma. De esta forma, la funcionalidad asociada a cada componente de aplicación debe ser descrita evitando detalles de elementos que sean privativos de algún tipo concreto de plataforma HW o SW. De la misma forma, los modelos de comunicación deben describir los detalles de comportamiento de dicha comunicación (cómo debe comportarse) y no características de implementación

En consecuencia, la metodología propuesta acepta dos mecanismos para la descripción de la funcionalidad. En primer lugar, esta descripción puede hacerse mediante diagramas que describan los algoritmos a implementar en cada una de las funciones o procedimientos que contiene el componente. En segundo lugar, se puede asociar a los componentes códigos fuente o librerías con la funcionalidad de dichas funciones o procedimientos. En ambos casos, los diagramas o códigos deberán abstenerse de incluir llamadas a servicios proporcionados por la plataforma, especialmente aquellos centrados en la implementación de comunicaciones o creación de tareas (mediante hilos o procesos). En su lugar, los detalles de creación, comunicación o cualquier otro tipo de relación con

otras tareas deben describirse explícitamente en el modelo, estando asociadas a los elementos de comunicación.

Para modelar la comunicación, se ha usado dos enfoques. El principal enfoque usado se basa en el paradigma cliente/servidor, donde los componentes (que actúan como clientes) llaman a servicios de otros componentes (que actúan como servidores). Los componentes pueden actuar como clientes y servidores al mismo tiempo, dependiendo de los servicios que requieran/proporcionen, ya que las llamadas de servicio pueden estar encadenadas. Estos servicios se definen a través de interfaces donde se agrupan los procedimientos y funciones internos al componente que pueden ser accedidos desde el exterior o aquellos que son requeridos por los procedimientos internos y se encuentran en otros componentes de aplicación.

Un segundo enfoque, usado en menor medida, es el basado en flujos de datos entre los componentes de la aplicación, definiendo una estructura concurrente en base a una estructura de "pipeline".

Para describir los detalles de la comunicación entre componentes, a través de sus interfaces, se utilizan canales. Estos canales actúan como medios de comunicación que conectan los componentes y por los cuales fluyen los datos intercambiados.

Cada uno de estos tres agentes (componentes de aplicación, interfaces y canales) tiene asociados un conjunto de propiedades que definen el comportamiento dinámico de la aplicación, el cual puede inferirse de dichas propiedades.

En los componentes de aplicación se puede definir parámetros como la cantidad máxima de llamadas simultáneas a servicios que proporciona el propio componente, y que pueden ser atendidas de forma concurrente.

Los interfaces capturan comportamientos concurrentes muy diferentes, con propiedades específicas: secuencial (solo una invocación a un servicio puede darse a la vez), protegida (múltiples invocaciones a un servicio pueden tener lugar, pero únicamente una puede comenzar), y concurrente (múltiples invocaciones a un servicio pueden tener lugar y pueden ejecutarse todas ellas a la vez).

Finalmente, los canales poseen un conjunto de propiedades con impacto en la semántica de comportamiento en los componentes de aplicación conectados por dicho canal. Estas propiedades describen detalles como que las llamadas sean bloqueantes o no bloqueantes, capacidad de almacenamiento (si las llamadas acumuladas sobrepasan esta capacidad, toda esa información se pierde, generando potenciales riesgos al sistema) o la

capacidad de poder asignar prioridad al canal, con lo que las llamadas a función transmitidas por ese canal heredaran dicha prioridad, estableciéndose un orden, por parte del componente de aplicación, a la hora de atender dichas llamadas a sus servicios proporcionados.

Como consecuencia, cuando un cliente llama a un servicio, se pueden dar tres escenarios: "espera completa", el cliente espera hasta la finalización del servicio; "espera parcial" el cliente espera hasta que el servidor atienda la llamada al servicio; "no espera", la ejecución no se detiene y continúa.

En este escenario, los servicios pueden proporcionar datos de vuelta requeridos por el cliente. En este caso, es necesario esperar hasta la finalización del servicio, bloqueando el flujo de ejecución del cliente. Este tipo de servicios evitan la creación de concurrencia entre el cliente y el servidor.

Estas propiedades de los canales combinadas con atributos de los componentes de aplicación y las interfaces definen la semántica del flujo de ejecución de la aplicación, pudiendo abarcar flujos secuenciales, con estructura de “pipeline”, concurrentes…, y, por tanto, un funcionamiento diferente que influye directamente en el rendimiento del sistema, como se mostrará en esta tesis (sección *Canales y Concurrencia* del capítulo *Ejemplos de Uso y Resultados*). Esta concurrencia dinámica, inferida de los diferentes valores de las propiedades antes mencionadas, se diferencia de la concurrencia estática. Esta última, consiste en hilos de ejecución persistentes a lo largo de la ejecución; un ejemplo claro de esto es cuando se denota que un componente es el “main” de la aplicación.

Además de esto, este modelo también incluye la definición de la estructura de espacios de memoria, usada para agrupar funcionalidad y permitir un mapeo específico de la funcionalidad en los recursos disponibles en la plataforma destino.

Como resumen, en este modelo se incluyen:

- Tipos de datos: estos se definirán para poder especificar los parámetros de las funciones descritas en los componentes. Estas funciones pueden estar definidas en el contexto de un interfaz o ser funciones internas de un componente.
- Interfaces: engloban la funcionalidad que es ofrecida/requerida por cada componente. Pueden tener asociado un conjunto de propiedades que definen el comportamiento de las llamadas a esas funciones.

- Puertos: representan una instancia de interfaz asociada a un componente de aplicación. Cuando un componente de aplicación proporciona o requiere los servicios de una interfaz, es necesario crear una instancia de interfaz en dicho componente, que se llama puerto. De esta forma, para poder realizar una comunicación es necesario que el componente cliente y el componente servidor contengan puertos con la misma interfaz, siendo una provista y otra requerida.
- Componentes de aplicación: son elementos que representan una cierta funcionalidad que, por sus características o por una decisión de diseño, ha sido agrupada formando una entidad. Estos elementos se comunican a través de puertos donde se especifican las interfaces que ofrecen y/o requieren. Está comunicación se lleva a cabo mediante canales. Estos componentes pueden ser activos, esto es, poseer su propio hilo de ejecución, o ser pasivos cuando no lo poseen.
- Funcionalidad de los componentes de aplicación: puede ser modelada de dos formas diferentes. El primero de ellos consiste en la especificación de los archivos funcionales, representando el conjunto de archivos que implementan la funcionalidad de un cierto componente en forma de código implementado en algún lenguaje de programación. En el modelo son asociados a cada uno de los componentes de la aplicación (y del entorno). En un segundo escenario, este comportamiento se modelo mediante diagramas de actividad, donde se capturan el envío/recepción de datos, así como las funciones internas del componente.
- Canales de comunicación: son los mecanismos usados para el transporte de información entre los componentes de aplicación.
- Estructura de la aplicación: se conforma en base a instancias de componentes de aplicación conectados por canales, a través de sus puertos. Para definir esta estructura se ha seguido el principio de separación entre comunicación y cómputo; la semántica de comunicación viene definida por los canales y la computación en los componentes.
- Espacios de memoria: Son elementos que se usan para agrupar funcionalidad, permitiendo que se comparta de manera más eficiente información entre los componentes de aplicación asociados a un mismo espacio de memoria. Esto también permite una optimización de las comunicaciones entre estos componentes a la hora de su implementación. Finalmente, el mapeo en plataformas heterogéneas es más factible si se agrupa la funcionalidad en bloques.
- Entorno del sistema. El entorno tiene dos interpretaciones dependiendo de la etapa de diseño en que nos encontremos. La primera de ellas es que el entorno se comporta como un agente abstracto que se usa para testear el sistema; se especifican varios escenarios de aplicación y, mediante el envío de estímulos al sistema, se analiza el comportamiento de este. En el otro escenario, el entorno se compone de los

elementos reales que van a interactuar con el sistema: si el sistema hace uso de un periférico, por ejemplo de una cámara, durante el proceso de prototipado se añade a la funcionalidad todos los drivers, archivos de código para su acceso y manejo, necesarios para su utilización.

1.2 Modelo de Descripción de Plataforma

Una parte esencial de los sistemas empotrados es la plataforma donde la aplicación se va a ejecutar. Como un primer paso, se ha de definir los diferentes componentes HW y SW, a partir de los cuales, se va a especificar dicha plataforma destino.

Los componentes HW abarcan los recursos de cómputo, como los procesadores, así como los diferentes tipos de elementos de almacenamiento de información (caches, memorias RAM y ROM), y los mecanismos de conexión (buses). También es posible encontrar otros componentes HW como los I/O, FPGAs…

Respecto a los componentes SW, estos especifican sistemas operativos y los controladores de dispositivos con el fin de permitir en uso de algún recurso HW específico.

Este modelo también incluye toda aquella información necesaria para poder realizar una síntesis de SW acorde con los recursos de cómputo presentes en la plataforma; información como compiladores y opciones de compilación y enlazado de estos.

Por tanto, en este modelo se incluyen:

- Componentes HW/SW: son los elementos que se van usar para la descripción de la plataforma. Ejemplos de componentes HW son el procesador, el bus, la memoria, una zona de lógica programable FPGA… Ejemplos de componentes de la plataforma SW son el sistema operativo o los controladores de dispositivo.
- Compiladores e información de compilación: es información necesaria para automatizar diversas etapas del proceso de diseño desde el modelo. Dan información sobre qué compiladores y opciones de los mismos se han de usar para la implementación del sistema.

1.3 Modelo Específico de Plataforma

Una vez que se han definido los componentes HW y SW, estos se usan para especificar la arquitectura de la plataforma destino. A su vez, los espacios de memoria definidos con anterioridad se mapean en los diferentes recursos de cómputo presentes en la plataforma.

Por tanto, este modelo incluye:

- Estructura de la plataforma: con instancias de los componentes HW y SW de plataforma interconectados se define la arquitectura de la plataforma.
- Mapeo arquitectural: una vez que se ha definido la arquitectura de la plataforma, los espacios de memoria son mapeados a los diferentes recursos de cómputo presentes en dicha plataforma.

Además de estos estos aspectos, en caso de sistemas en red, este modelo incluye la especificación de los nodos que van a componer nuestro sistema distribuido. También incluye la especificación de dicho sistema en base a la interconexión de los nodos previamente definidos.

Este modelo también incluye:

- La especificación de los nodos, así como su estructura interna.
- El sistema distribuido compuesto de nodos interconectados.

1.4 Análisis y exploración de alternativas

Además de capturar los diferentes aspectos estructurales del sistema, también al modelo se le puede añadir información adicional que permita realizar una batería de análisis de dicho diseño. Para ello se definen otros aspectos del modelado que son transversales a los modelos previamente presentados.

Para realizar una implementación o modelo ejecutable del sistema, cada componente (de aplicación, HW o SW) del modelo tiene asociado un conjunto específico de propiedades, con unos valores fijos. Sin embargo, como se presentó en la sección *Análisis* del capítulo *Metodología de Modelo Único,* durante el proceso de diseño es necesario explorar el impacto en el rendimiento del sistema de diversas alternativas, atendiendo a diferentes valores que pueden tomar diversas propiedades de los

componentes. Así, este conjunto de valores de diferentes variables conforma el espacio de diseño que se va a poder explorar.

Para definir el conjunto de alternativas a explorar, estas variables de diseño se han de especificar de una manera más genérica, capturando todos los potenciales valores que se quieren analizar, en lugar de un único valor. De la misma manera, también se pueden relacionar dos o más variables de forma que únicamente un subconjunto de todas las combinaciones de los valores que pueden tomar dichas variables va a ser analizado.

En otra situación, se pueden plantear escenarios de comportamiento del sistema, donde se analicen propiedades como si el sistema es o no planificable, o si una cierta métrica del sistema se cumple o no, en función de una cierta configuración del mismo. Estos escenarios de análisis requieren de nuevos elementos expresivos de MARTE.

Para soportar todas estas alternativas de diseño, la metodología incluye:

- Variables de diseño: definen el espacio de diseño a explorar. Estas variables pueden ser desde el número de procesadores presentes en nuestra plataforma, hasta tamaños de memorias y velocidades de transmisión de buses, hasta semántica de canales o número máximo de hilos concurrentes que se pueden dar por componente de aplicación. Estas variables de diseño se emplean en el modelo en lugar de los valores fijos, al principio del proceso de diseño. Una vez que se eligen los valores finales para cada una de las variables de diseño a través de las distintas etapas de exploración y análisis, se reemplazan en el modelo dichas variables de diseño iniciales por sus valores finales.
- Restricciones al diseño: son reglas que se usan para establecer relaciones entre diferentes variables de diseño. De esta forma, se pueden establecer reglas entre propiedades de diferentes componentes. Por ejemplo, si puedo tener 2,4 u 8 procesadores en la plataforma, sus caches de datos no pueden ser mayores de 2 MB.
- Requisitos no-funcionales: son condiciones del diseño que han de cumplirse y que sirven para que, durante el proceso completo de implementación, se vayan validando las configuraciones generadas, para comprobar si nuestros diseños van cumpliendo los requisitos o no.

2. Estructura del Modelo

La organización de los tres sub-modelos (PIM, PDM y PSM), anteriormente descritos en la Figura 10, es demasiado amplia y genérica como para definir completamente una metodología de diseño en cuanto a su estructuración. Tomando como ejemplo el PIM, este incluye todos los aspectos que caracterizan a la aplicación, que son muchos y muy variados. Por tanto, limitarse únicamente a la organización en estos sub-modelos no es la solución adecuada para grandes sistemas, ya que la cantidad de información que se ha de tener en cuenta es muy amplia. El hecho de considerar como núcleo del entorno de desarrollo un modelo único de alto nivel implica que dicho modelo ha de capturar una gran cantidad información, requerida en diferentes etapas de diseño, así como de permitir el uso e integración de diferentes herramientas para tales etapas. Este hecho añade problemas adicionales al propio modelado del sistema que este trabajo va a resolver.

Para abordar el manejo de toda esta información y dotar de organización al modelo, la metodología de modelado se estructura en base a vistas. Cada vista contiene el modelado de un aspecto específico del sistema. Continuando con el ejemplo del PIM, este se compone de varios aspectos como los componentes de aplicación, los canales, las interfaces… Estos aspectos del PIM se capturan en diferentes vistas del modelo.

Por lo tanto, la metodología UML/MARTE presentada en esta tesis está basada en la separación y organización de los diferentes aspectos del sistema como una solución para crear modelos de sistemas complejos, con el fin de facilitar su desarrollo. Esta organización se basa en vistas del modelo, estructuradas de forma que cada una captura un aspecto específico del sistema a diseñar. La idea original de la separación en vistas surgió dentro del contexto del proyecto COMPLEX, de las personas F. Ferrero y F. Herrera. A lo largo de otros proyectos estás fueron modificadas y se añadieron nuevas. Todo esto se citará explícitamente en las siguientes secciones.

El conjunto de vistas que se incluyen en esta metodología se muestra en la Figura 11. En dicha figura se pueden distinguir dos categorías; la primera, resaltada en color azul, denota las vistas que se deben usar con el fin de diseñar un sistema empotrado. La segunda, destacada en color verde, son las vistas complementarias necesarias para definir un sistema distribuido, compuesto de nodos interconectados, conformando una red de sistemas.

En las subsiguientes subsecciones se describen cada una de estas vistas, detallando qué aspectos del sistema se incluyen en ellas. Cada vista se englobará en el

correspondiente sub-modelo (PIM, PDM, PSM) de acuerdo a los aspectos del modelo que se incluyen en ella.

Hay, no obstante, una excepción a esta regla de asociación de vistas a sub-modelos. La vista de verificación es general a todo el proceso de diseño, y por tanto se incluye en todos los sub-modelos.

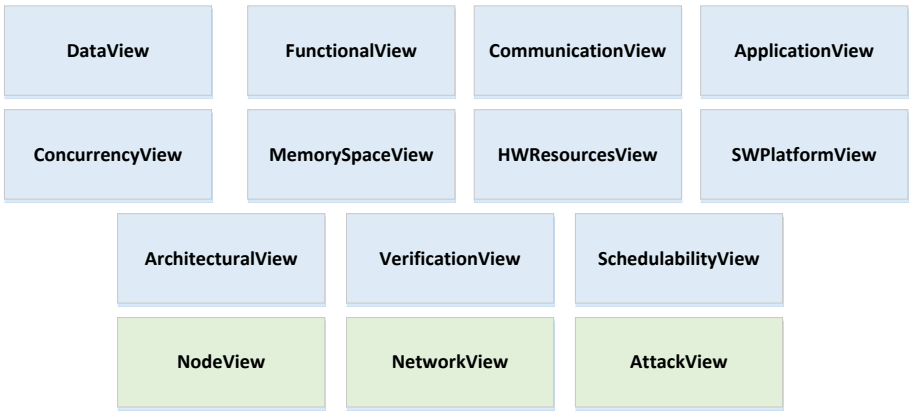


Figura 11 Conjunto de vistas definidas en esta metodología

2.1 Vistas: Modelo Independiente de Plataforma

El PIM incluye todo aquello que involucra el modelado de la aplicación, independientemente de la plataforma en la que posteriormente vaya a ser implementado. Así la aplicación se compone de componentes, estos tienen interfaces, las interfaces se componen de funciones, que poseen argumentos definidos conforme a tipos de datos y que tienen asociados elementos que modelan su comportamiento o que especifican detalles que se utilizarán para la implementación de dicho comportamiento. Además de esto, incluye la estructura en espacios de memoria de la aplicación para facilitar el mapeo en la plataforma.

Para poder describir todos estos elementos, el PIM se divide en las vistas siguientes:

- Vista de Datos
- Vista Funcional
- Vista de Comunicación
- Vista de Aplicación

- Vista de Concurrencia
- Vista de Espacios de Memoria
- Vista de Verificación

Vista de Datos ("DataView")

En esta vista se definen los tipos de datos que van a ser necesarios para especificar la funcionalidad del sistema. Los tipos de datos se usan principalmente con el fin de especificar los parámetros de los servicios agrupados en interfaces.

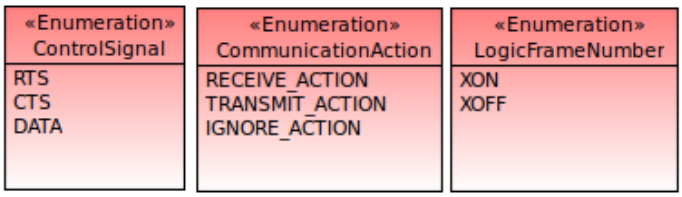


Figura 12 Definición de datos del tipo enumeración

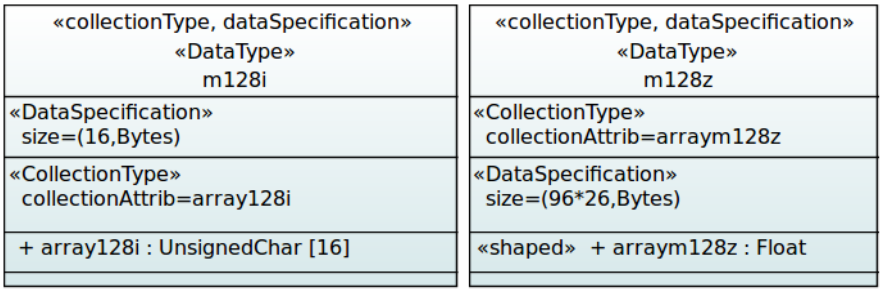


Figura 13 Definición de datos del tipo array

Los ejemplo de las Figura 12 y Figura 13 muestran dos ejemplos de tipos de datos que se incluyen en esta vista: enumeraciones y arrays.

Para más detalles [oo)].

Inicialmente definida por F. Ferrero y F. Herrera. La mayor parte de cómo definir los diferentes tipos de datos fue realizada por mí.

Vista Funcional ("FunctionalView")

Esta vista incluye:

- Interfaces
- Archivos
- Librerías
- Carpetas de archivos

Esta vista define la funcionalidad requerida/proporcionada por los componentes de aplicación para intercambiar datos. Esta funcionalidad se encapsula en interfaces que son proporcionados/requeridos por los componentes de la aplicación. Esta funcionalidad es modelada como métodos, que incluyen sus argumentos en términos de tamaño y tipo de dato (previamente definidos en la "DataView"). La Figura 14 muestra un ejemplo de interfaz donde se puede ver dos de estos métodos, con sus correspondientes parámetros.

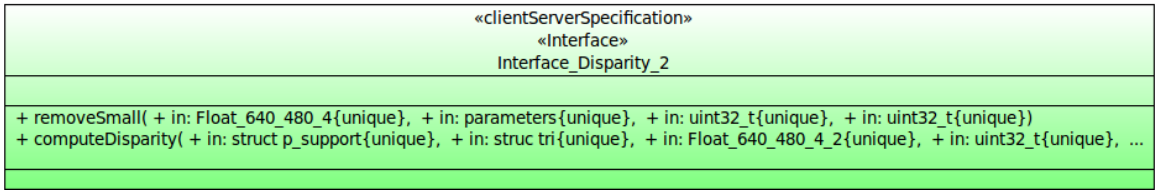


Figura 14 Interfaz

También se incluyen la definición de los archivos que implementan la funcionalidad de cada uno de los componentes de aplicación (Figura 15), así como otros aspectos funcionales como librerías, carpetas auxiliares del código funcional, que serán necesarias en el proceso de compilación.

Figura 15 Modelado de archivos

Estos elementos van a ser necesarios para poder realizar la compilación de la aplicación y obtener los correspondientes ejecutables durante la generación de modelos ejecutables o prototipos del sistema.

Más detalles en [oo)] y [rr)].

Inicialmente definida por F. Ferrero y F. Herrera. Yo añadí que en ella se incluyeran el modelado de los archivos, librerías… También simplifique el modelado de las interfaces.

Vista de Comunicación ("CommunicationView")

En esta vista se capturan los diferentes tipos de canales de comunicación que van a definir la semántica en el intercambio de datos entre de los diferentes componentes de la aplicación.

La semántica de cada tipo de canal viene defina por una serie de atributos. Un primer conjunto de atributos permite definir si la comunicación es bloqueante. Este bloqueo se puede dar en dos situaciones. En el primer caso, el componente de aplicación que hace una llamada a un servicio proporcionado por otro componente, se bloquea hasta que el componente que ofrece dicho servicio atiende dicha petición. También se puede dar la situación que el componente de aplicación solicitante de un servicio se quede bloqueado hasta que esta petición sea almacenada en el canal, si dicho canal tiene la característica de poseer capacidad de almacenar llamadas. El segundo caso de bloqueo consiste en que la aplicación llamante se bloquea esperando que la función llamada le devuelva datos.

También se puede dar prioridad a un canal para definir el orden por el cual el componente de aplicación que ofrece servicios, atiende llamadas entrantes por diferentes canales. De esta forma se define una jerarquía en las comunicaciones que posee un componente cuando ofrece servicios de interfaz.

Otro atributo define el tiempo máximo por el cual el cliente espera para la respuesta de la función llamada.

Finalmente, el último atributo especifica si las llamadas múltiples a una misma función transmitidas a través del canal tienen que ser sincronizadas al retorno de las mismas y transmitirse como un conjunto ordenado.

Además de tipos de canales, otros mecanismos de comunicación, como variables compartidas, también pueden ser modelados e incluidos en esta vista.

Más detalles en [oo)], [rr)] y [ss)].

Definida por mí (en la versión inicial de F. Ferrero y F. Herrera existía la vista de "Communication&Application"; esta se separó como consecuencia del trabajo desarrollado en esta tesis).

Vista de Aplicación ("ApplicationView")

Esta vista incluye los componentes que van a conformar nuestra aplicación. Estos componentes se especifican mediante:

- Sus puertos donde se asocian las interfaces que denotan los servicios que proporcionan/requieren.
- Se le asocian los archivos que representan la implementación de su funcionalidad (Figura 16).
- Se asocian librerías y carpetas de archivos auxiliares necesarios para compilar el componente.

Esta vista también incluye la definición estructural de la aplicación. Esta estructura es creada mediante instancias de componentes de aplicación conectadas por canales a través de los puertos de dichos componentes. Las propiedades de estos canales están definidas por los tipos de canal incluidos en la vista de comunicación.

Más detalles en [oo)], [rr)] y [ss)].

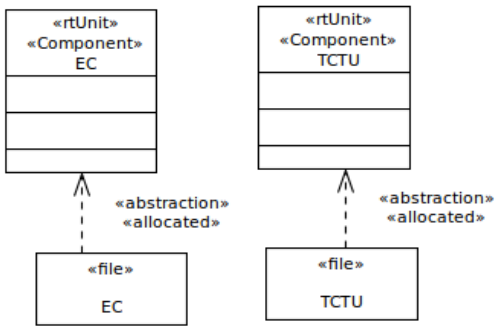


Figura 16 Asociación de archivos a componentes de aplicación

Definida por mí (en la versión de F. Ferrero y F. Herrera existía la vista de "Communication&Application", esta se separó como consecuencia del trabajo desarrollado en esta tesis).

Vista de Concurrencia ("ConcurrencyView")

En esta vista se definen propiedades asociadas a las tareas concurrentes de la aplicación. Utilizando esta vista, las tareas pueden ser caracterizadas por una prioridad y permiten una asignación específica al recurso de la plataforma donde se quiere que se ejecute dichas tareas. Esta información es especialmente relevante si estas tareas van a ser analizadas en un análisis de planificabilidad.

Más detalles en [ss)].

Vista definida por mí.

Vista de Espacios de Memoria ("MemorySpaceView")

En esta vista se modelan los diferentes espacios de memoria donde los diferentes componentes de aplicación son mapeados, modelando los procesos del sistema. De esta forma se agrupa componentes que comparten memoria, permitiendo una mayor flexibilidad a la hora de asociar funcionalidad a recursos de plataforma.

Más detalles en [oo)] y en [rr)].

Vista definida por mí.

Vista de Verificación ("VerificationView")

En esta vista se modela el entorno del sistema. El entorno del sistema debe ser considerado a lo largo del proceso de diseño ya sea para un proceso de simulación, considerando los estímulos que recibirán el sistema y cómo tiene que reaccionar ante ellos, actuando como un banco de pruebas de la aplicación. Dado que parte de la verificación del sistema está relacionada exclusivamente con la funcionalidad independiente de plataforma, esta vista contiene información asociada al PIM. No obstante, como se dijo anteriormente, las actividades de verificación cubren todo el proceso de diseño, y por tanto esta vista no es exclusiva de este sub-modelo sino compartida por los tres sub-modelos.

Más detalles en [oo)] y [k)].

Vista definida por mí.

2.2 Vistas: Modelo de Descripción de Plataforma

El PDM incluye todos los componentes de plataforma que van a ser usados para la descripción de la arquitectura de la plataforma HW/SW. Las vistas que se incluyen son:

- Vista de Recursos HW
- Vista de Recursos SW

Además de estas, en este modelo se pueden incluir dos nuevas vistas. Estas tienen que ver con el entorno del sistema en dos situaciones diferentes:

- Vista de Verificación
- Vista de Ataques

Vista de recursos HW ("HWResourcesView")

Esta vista incluye la especificación de los componentes HW que van a ser usados para definir la plataforma. Estos componentes HW pueden ser procesadores, buses, memorias… Atributos como frecuencia, tamaño de memoria, ancho de banda…, capturan las propiedades más representativas de dichos elementos HW.

Más detalles en [oo)].

Inicialmente definida por F. Ferrero y F. Herrera. Los diferentes elementos de MARTE usados para modelar los elementos HW fueron seleccionados por mí (con el asesoramiento de J. Medina), así como los atributos y diferentes mecanismo expresivos usados para poder capturar el conjunto de propiedades que los caracterizan [oo)]. Estos se fueron incrementando a lo largo de los diferentes proyectos.

Vista de recursos SW ("SWPlatformView")

En esta vista se definen los componentes SW que van a ser usados en la especificación de la plataforma. Estos recursos SW pueden ser diferentes tipos de sistemas operativos o los controladores de dispositivos requeridos con el fin de permitir el uso de algún recurso HW específico como puede ser un periférico.

Más detalles en [oo)].

Inicialmente definida por F. Ferrero y F. Herrera. Los diferentes elementos de MARTE usados para modelar los elementos SW fueron seleccionados por mí (con el

asesoramiento de J. Medina), así como los atributos y diferentes mecanismo expresivos usados para poder capturar el conjunto de propiedades que los caracterizan [oo)]. Estos se fueron incrementando a lo largo de los diferentes proyectos.

Vista de Verificación ("VerificationView")

En la sección anterior *Vistas: Modelo Independiente de Plataforma* ya se incluyó esta vista, como lugar donde se capturaban escenarios de validación de la aplicación. En este caso, en esta vista también se usa para hacer una descripción del entorno, pero donde se incluya lo necesario para implementar la funcionalidad para acceder a un elemento específico de dicho entorno (por ejemplo, una cámara) y no como un modelo para la verificación funcional de la aplicación.

Más detalles en [oo)] y [k)].

Vista definida por mí. Toda la metodología del entorno es descrita en la sección *Modelado del Entorno* del capítulo *Modelado de Sistemas Electrónicos.*

Vista de Ataques ("AttackView")

En esta vista se definen un conjunto de agentes externos cuya acción es intentar crear en nuestra red problemas con el fin de hacer que las comunicaciones entre los nodos se rompan. Este conjunto de atacantes permite establecer escenarios de análisis que tengan como fin estudiar la robustez de la red contra ataques maliciosos externos.

Más detalles en [a)] y [pp)].

Vista definida por mí. Vista definida por mí. Toda la metodología del entorno es descrita en la sección *Modelado del Entorno* del capítulo *Modelado de Sistemas Electrónicos.*

2.3 Vistas: Modelo Específico de Plataforma para un Sistema Empotrado

En el PSM se describe la arquitectura de la plataforma HW/SW y la asignación de la aplicación (espacios de memoria) a los diferentes recursos de dicha plataforma. También se puede incluir en este sub-modelo todos los elementos expresivos necesarios para definir escenarios donde se analice la planificabilidad del sistema, ya que se asocian

a recursos de cómputo de la plataforma las funciones de la aplicación, caracterizadas con propiedades temporales como tiempos de ejecución.

Las vistas que se incluyen son:

- Vista Arquitectural
- Vista de Planificabilidad

Vista Arquitectural ("ArchitecturalView")

La vista incluye la especificación de la arquitectura de la plataforma de nuestro sistema empotrado. Dicha plataforma es creada en base a instancias de componentes HW y SW previamente modelados en las vistas "HWResourcesView" y "SWPlatformView". Una vez que la arquitectura de la plataforma HW/SW está definida, se realiza el mapeo de la aplicación (espacios de memoria o componentes de aplicación) en los diferentes recursos HW/SW.

Más detalles en [oo)].

Inicialmente definida por F. Ferrero y F. Herrera.

Vista de Planificabilidad ("SchedulabilityView")

Esta vista incluye la definición de modelos de análisis de planificabilidad. Basándose en trabajos como en [6] y [7], los diferentes elementos del modelo han sido enriquecidos con la información necesaria para capturar propiedades de tiempo real, como plazos de ejecución o tiempos de ejecución, y que permiten realizar, en el entorno de desarrollo presentado en esta tesis, este tipo de análisis.

Más detalles en [ss)]. Vista definida por mí. Toda la metodología del entorno es descrita en la sección *Modelado del Entorno* del capítulo *Modelado de Sistemas Electrónicos.*

Vista definida por mí.

2.4 Vistas: Modelo Específico de Plataforma para Sistemas Distribuidos

El PSM para los sistemas distribuidos se compone principalmente de dos vistas. La primera de ellas incluye la especificación de los nodos. Estos nodos, en sí mismos, son elementos con su propia plataforma HW/SW, con lo que pueden ser vistos como un sistema a ser modelado como en la "ArchitecturalView" como si fueran un sistema empotrado. Dependiendo del escenario, también en la propia definición del nodo se puede incluir la aplicación que ejecutan [a)].

A partir de instancias de estos componentes nodo, se conforma el sistema distribuido.

Las vistas que se incluyen son:

- Vista de Nodos
- Vista de Red

Vista de Nodos ("NodeView")

Los componentes nodo pueden ser elementos estructurados como se muestra en la Figura 17 y Figura 18. En este caso y utilizando los elementos del modelo definidos en la "ApplicationView", "HWResourcesView" y "SWPlatformView la estructura interna del nodo es definida, en lo relativo a la plataforma HW/SW y al mapeo de la aplicación.

En otro escenario, el componente nodo es una caja negra de la que solo se caracteriza parámetros para especificar la comunicación, como por ejemplo potencia de transmisión, número de intentos para realizar el envío de datos…

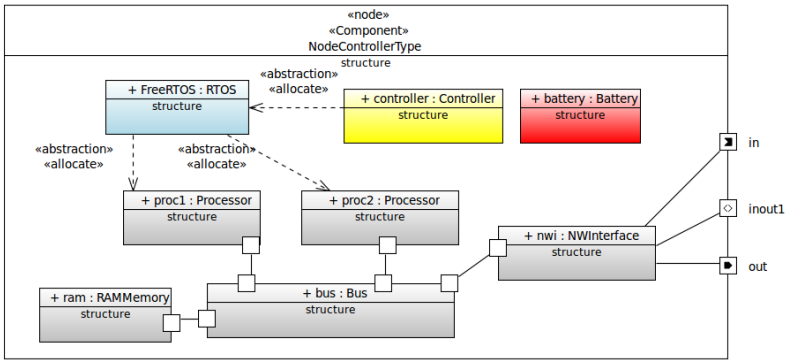


Figura 17 Estructura de un componente nodo del tipo "Controller"

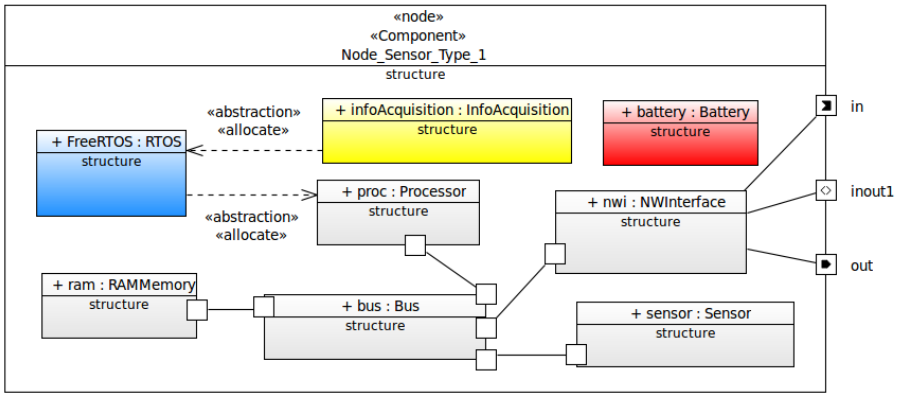


Figura 18 Estructura HW/SW y con aplicación de un componente nodo del tipo sensor

En otro escenario, la estructura del nodo solo incluye la definición de la plataforma HW/SW (Figura 19), dejando el mapeo funcional cuando se defina la red en la Vista de Red.

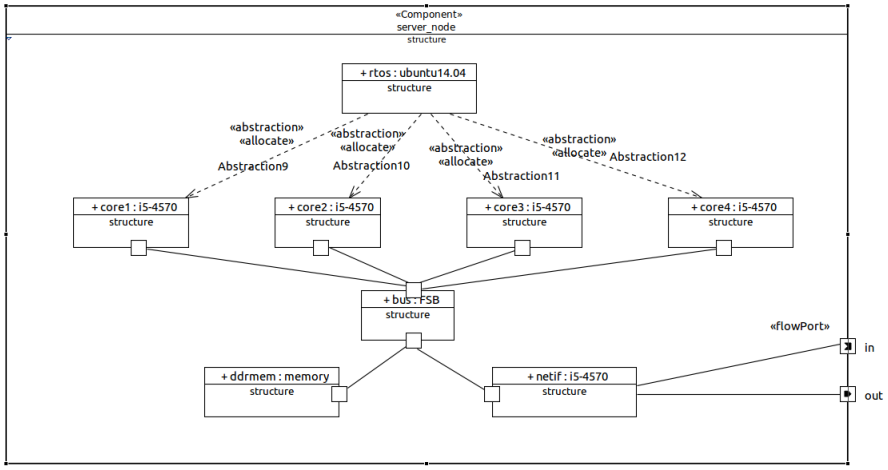


Figura 19 Estructura HW/SW de un nodo

Más detalles en [a)] y [pp)].

Vista definida por mí.

Vista de Red ("NetworkView")

En esta vista se especifica el sistema distribuido, compuesto por instancias de componentes nodo interconectados. El tipo de red modelada dependerá del cómo son los nodos que la componen. Uno de los casos que en esta tesis se presentará más en detalle es de la una red de sensores (sección *Redes de sensores* de este capítulo). Otro tipo de

sistema distribuido (descritos en [pp)]) son aquellos en que la plataforma HW está compuesta de nodos descritos como si fueran sistemas empotrados pero en los que únicamente la plataforma HW está definida. Luego, un sistema operativo común se mapea a ellos, con lo que esos nodos son manejados por dicho sistema operativo, que se encarga de distribuir, sobre dichos nodos, los componentes de aplicación mapeados.

Otro escenario es aquel en el cual los nodos están compuestos de plataforma HW/SW completa, mapeando los componentes de aplicación en dichos nodos.

En todos los casos, los nodos están conectados entre sí mediante una red, por ejemplo Ethernet.

Con estos ejemplos anteriores se ilustran una serie de ejemplos específico. En el caso general, la metodología permite la especificación de un sistema de sistemas, donde la estructura y detalle pueden variar a necesidad del diseñador.

Más detalles en [a)] y [pp)].

Vista definida por mí.

3. Especificación del sistema: aplicación

Una vez presentada la estructura general del modelo, es momento de definir los elementos que permitirán describir la información incluida en cada vista. Además, es necesario definir cómo han de usarse para permitir la utilización del modelo de alto nivel en las subsiguientes etapas del proceso de diseño. Para cubrir todos estos detalles se analizará cada parte del modelo por separado, comenzando por la aplicación.

Como punto de partida general, el diseñador puede comenzar la especificación de su funcionalidad mediante una descripción en pseudo-código del algoritmo a implementar. Luego, tomará decisiones sobre cómo lo implementará, si lo hará de forma que el flujo de ejecución sea secuencial o concurrente, en componentes o no. Otro escenario que se puede dar es que se tenga una versión implementada de una funcionalidad con un flujo secuencial y se quiera paralelizar con el fin de explorar las plataformas con múltiples recursos de cómputo, esto es, analizar qué impacto tiene en el rendimiento del sistema la selección de una u otra plataforma en función de los recursos HW/SW que tienen disponibles dichas plataformas. En cualquier escenario que nos encontremos, la concurrencia, como decisión de diseño, presenta grandes retos a abordar.

El diseño de aplicaciones concurrentes seguras y eficientes debe abordar el correcto acceso a datos compartidos, el sincronismo entre tareas y los inter-bloqueos entre elementos concurrentes, entre otras cosas.

En este contexto y con el fin de abordar la problemática de los sistemas concurrentes, la aplicación de formalismos matemáticos es una elección muy extendida entre la comunidad científica. Propuestas como [8] son buenos ejemplos de cómo los formalismos matemáticos son herramientas adecuadas para poder detectar y solventar los problemas de consistencia y funcionamiento en aquellos sistemas donde la concurrencia está presenta.

Los formalismos matemáticos aplicados a sistemas concurrentes proporcionan reglas en el diseño. Estas reglas no deben entenderse como restricciones en su amplio sentido de la palabra. Estas reglas pueden entenderse como recomendaciones que aseguran el correcto funcionamiento del sistema, a nivel funcional y que, si no se respetan, nada asegura, por construcción, dicho funcionamiento. Por tanto, considerando estas reglas de diseño basadas en dichos formalismos matemáticos, el sistema especificado es correcto por construcción, entendiendo correcto como la ausencia de indeterminismo en la forma de bloqueos, inanición, carreras críticas…

En una primera aproximación, la aplicación y la comunicación se hicieron con elementos del capítulo de MARTE GRM presentado en la sección *Paquete de Fundamentos* del capítulo *Estado del Arte*. De esta forma la concepción del sistema se hacía de forma genérica, únicamente enfocada a la definición de los elementos concurrentes y de los mecanismos de comunicación de datos. Esto es descrito en la siguiente sección *Formalización de modelos UML/MARTE*.

Basándose en el trabajo de esa sección, la metodología se amplío, usando elementos del capítulo de MARTE HLAM presentado en la sección *Paquete de Diseño* del capítulo *Estado del Arte*. Las dos formas de modelar la funcionalidad y la comunicación se basan en los mismos principios semánticos que se explicarán en la siguiente sección *Formalización de modelos UML/MARTE*. Sin embargo, este hecho no ha de verse como dos metodologías diferentes: son compatibles entre ellas y con objetivos diferentes cada una de ellas.

3.1 Formalización de UML/MARTE: ForSyDe

Con el fin de poder abordar el modelado de los sistemas concurrentes de una forma metódica y con la mira puesta en generar sistemas deterministas, se utilizó el formalismo ForSyDe (sección *ForSyDe* de la sección *Lenguajes* del capítulo *Estado del Arte*). Usando ForSyDe, se ha explorado el desarrollo de un marco donde los modelos UML/MARTE son formalmente representados. Este trabajo tomó como base los fundamentos formales realizados por F. Herrera, E. Villar presentados en [33], llevándose a cabo con la colaboración de E. Villar, F. Herrera y J. Medina.

A partir de un subconjunto de conceptos de UML y de MARTE, el diseñador puede concebir el sistema como un conjunto de elementos concurrentes comunicándose entre sí. La correspondencia entre los elementos UML/MARTE seleccionados y el metamodelo ForSyDe permite crear modelos que capturan características formales esenciales, generando modelos semánticamente bien definidos y unívocos.

Como se mostró en la sección *ForSyDe* del estado del arte, este formalismo está compuesto de dos elementos: señales y procesos. Las señales representan los datos que se transmiten entre los procesos, que computan dichos datos para generar unos nuevos. Esto hace que las actividades de transmisión de datos y de cómputo estén claramente diferenciadas unas de otras.

Esta concepción de la comunicación y de la funcionalidad ha sido integrada en la metodología de modelado de esta tesis. Esto es de gran ayuda para poder explorar

mecanismos de comunicación con diferentes propiedades y ver sus prestaciones y cómo afectan al rendimiento del sistema. Extrapolado a la aplicación, los componentes de aplicación incluidos en el modelo de alto nivel deben contener funcionalidad pura, limitando los detalles de comunicaciones al uso de canales. Esto permite la explotación de diferentes semánticas de comunicación, con canales con diferentes propiedades. De la misma forma, se puede sustituir un componente por otro, con la única condición de que sus interfaces sean compatibles con las del componente sustituido.

El primer trabajo donde se abordó la relación UML/MARTE-ForSyDe en esta tesis fue en [m)]. En este trabajo, se establece una relación entre el elemento de cómputo (entendido en este trabajo como "concurrency resources", Figura 20) y los procesos ForSyDe (Figura 20). De la misma forma, los elementos de comunicación (entendidos en este trabajo como "communication media", Figura 20) que actúan como medios de transporte de información entre los elementos de cómputo son relacionados con las señales ForSyDe (Figura 20).

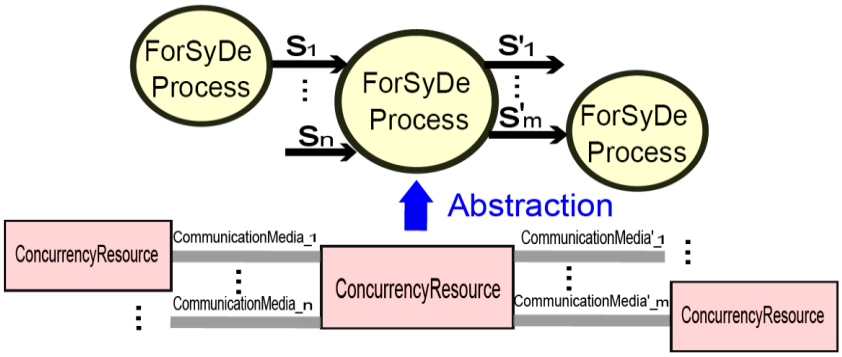


Figura 20 Representación entre elementos MARTE y ForSyDe.

De esta forma, la estructura de la funcionalidad a capturar en el modelado ha de ser construida en base a componentes de aplicación, que computan los datos que le son llegados por elementos de comunicación, cuya única función es dicho transporte de datos.

La formalización ForSyDe también tiene impacto en la especificación de la funcionalidad del elemento de cómputo; también se considera la formalización en la descripción del comportamiento del elemento de cómputo.

El comportamiento de un elemento concurrente es entendido como una máquina de estados finita, donde en cada una de los estados se recibe datos, se computan dichos datos y se obtienen unos nuevos.

La captura del comportamiento de los elementos concurrentes puede ser hecha mediante una máquina de estados UML explícita. Este diagrama está enfocado en capturar los estados que recorre el objeto durante su vida, así como el conjunto de condiciones que denotan la transición entre los estados. En UML, cada uno de estos estados tiene asociado una propiedad llamada "do", donde se puede incluir la descripción de la funcionalidad realizada en dicho estado. Esta descripción puede ser captura mediante un diagrama de actividad de UML.

El diagrama de actividad para modelar el comportamiento

El diagrama de actividad puede modelar, por sí mismo y de forma completa, el comportamiento del elemento concurrente. En este caso, no hay una clara identificación de los estados del elemento; los estado ejecutados son implícitos.

Los diagramas de actividad representan un conjunto de actividades individuales que han de ser ejecutadas de forma ordenada, de forma que se describe el comportamiento completo del elemento. Esas actividades pueden estar compuestas de acciones atómicas que representan diferentes funcionalidades, pudiendo estar relacionados con llamadas a métodos o descripciones algorítmicas. Cada una de estas actividades específicas es representada por la función de ForSyDe f_α.

En base a los principios definidos por E. Villar, la estructura genérica y básica del diagrama de actividad es mostrada en la Figura 21. Esta figura representa un estado interno que es representado en ForSyDe como ω_j. No es relevante si esta figura representa la funcionalidad de un estado de un diagrama de estados o es parte de un diagrama de actividad. Como aproximación general, en cada transición de estado, el elemento toma datos de sus entradas, los computa aplicando la funcionalidad específica del estado, y envía los correspondientes datos generados.

Como se mostró en la sección *ForSyDe* del estado del arte, un estado ω_j se componente de dos diferentes estados P_j y D_j. En este caso, P_j engloba la funcionalidad que se encuentra en dos secciones de espera (identificadas como la recepción de datos), por tanto entre dos secciones (*Receive_1… Receive_N*). Por tanto, P_j es representado como la estructura mostrada en la Figura 21.

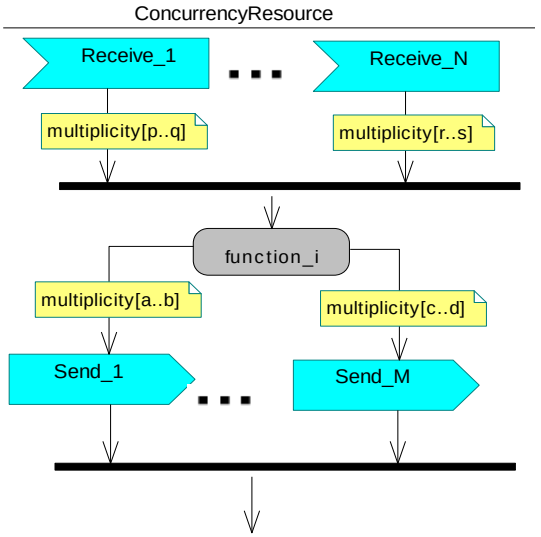


Figura 21 Formalización de la descripción de comportamiento de un elemento de cómputo

Para modelar la recepción y consumo de datos, la recepción de estos es modelado por "AccepEventAction". Este elemento de UML representa la llamada específica a un método que modela/representa la adquisición de nuevos datos o, simplemente, que un nuevo dato está disponible. En nuestro caso, estos elementos representan la llegada de datos a través de un "communication media".

Para modelar el envío de datos, se usan los elementos de UML "SendSignalAction". De nuevo, estas pueden representar la llamada un método o, simplemente, que un nuevo dato ha sido generado y está disponible.

Para realizar el cómputo, el elemento concurrente ha de tener disponibles todos los datos en las entradas del estado correspondiente. Para modelar esto, se pone en el modelo un elemento de sincronización de todas las salidas de los "AccepEventAction". Esto se hace mediante un "Join Node".

D_j expresa todos los valores internos que caracterizan el estado. Estos valores internos pueden representar aspectos relevantes que tienen que ser modelados de forma explícita en el modelo y que pueden cambiar dependiendo de cada estado interno.

También es necesario modelar el número de datos consumidos/generados en un estado concreto. Para ellos se utiliza el elemento de UML "Multiplicity". Esta valor representa el número mínimo y máximo de datos que pueden ser aceptados/generados en cada estado. El mínimo valor implica que no se puede disparar la ejecución de dicho

estado hasta que no se haya alcanzado dicho número de datos en una entrada dada. Para que se ejecute dicho estado, en todas las entradas han de alcanzarse ese valor mímino de datos.

Depende del estado estados S_j , se pueden consumir diferente cantidad de eventos. Esto se representa por la función ForSyDe $\upsilon(z)$:

$$v_1(z) = \gamma(\omega_j) = \begin{cases} p \\ \dots \\ q \end{cases} \qquad v_n(z) = \gamma(\omega_j) = \begin{cases} r \\ \dots \\ s \end{cases}$$

$$\forall\, z, j \;\in \mathbb{N}_0 \wedge \{p, q, r, s\} \;\in\; \mathbb{N}$$

En la Figura 21, la acción "function_i" representa la funcionalidad que el elemento concurrente realiza en el estado. Esta función calcula los datos de salida. La notación ForSyDe es:

$$lenght(\, f_j(a_1 \dots a_N), \omega_j) = \begin{cases} v'_1(z) = length(a'_1) \begin{cases} a \\ \dots \\ b \end{cases} \\ \dots \\ v'_M(z) = length(a'_M) \begin{cases} c \\ \dots \\ d \end{cases} \end{cases}$$

$$\forall\, z, j \;\in \mathbb{N}_0 \wedge \{p, q, r, s\} \;\in\; \mathbb{N}$$

Esta función también determina el número de datos generados en cada salida.

En esta forma de modelar la funcionalidad de un elemento concurrente es necesario considerar una restricción. El uso de nodos "fork" para modelar la concurrencia interna del elemento no está considerada. Como aproximación general, solo se puede considerar la concurrencia interna en el caso que las entradas y salidas de cada uno de los flujos internos concurrentes sean unívocas. Entre varios flujos concurrentes es esencial conocer de qué entradas son tomados los datos y por cuáles de las salidas son envidos los resultados; solamente un flujo concurrente puede acceder a un mecanismo de comunicación ("communication media").

La separación entre comunicación y cómputo es el principio en el cual se fundamenta todo el trabajo desarrollado en esta tesis en lo que especificación de la funcionalidad se refiere. El hecho de diferenciar elementos funcionales y de comunicación simplifica el estudio y el análisis de los sistemas concurrentes ya que las propiedades formales de cada uno de los elementos que conforman el sistema pueden ser obtenidas de manera más fácil.

En los trabajos [i)], [j)] y [m)] se muestran más detalles de esta relación UML/MARTE-ForSyDe.

Proyectos en los que este trabajo se desarrolló:

1. *SATURN, [l)] y [vv)]. Director principal: E. Villar; presupuesto: 193200€; periodo: 2008-2010.*

Publicaciones resultantes:

1. *Formal Foundations for the Generation of Heterogeneous Executable Specifications in SystemC from UML/MARTE Models, [i)].*
2. *Formal Support for Untimed MARTE-SystemC Interoperability, [j)].*
3. *Formal Modeling for UML/MARTE Concurrency Resources, [m)].*

3.2 Canales y Concurrencia

Tomando como principio de diseño la separación entre comunicación y cómputo, se desarrolló la metodología de modelado en lo que a la aplicación se refiere orientada a la realización de la síntesis de código.

Esta metodología presenta claras diferencias a la versión inicialmente realizada por F. Ferrero, F. Herrera y mía y presentada en [b)]. En esta, no había distinción entre comunicación y aplicación (de ahí que únicamente hubiera una única vista "Communication&ApplicationView").

Como evolución a la anterior versión de la metodología, se separaron los aspectos funcionales y de comunicación, siguiendo el mismo principio de lo desarrollado en el trabajo de formalización con ForSyDe, mostrado en la sección anterior, *Formalización de UML/MARTE: ForSyDe.*

Como consecuencia de esto se crearon las vistas "ApplicationView" y "CommunicationView", para hacer explícita la separación de funcionalidad y comunicación. De esta manera, el diseñador puede definir la estructura de su aplicación y luego, mediante diferentes tipos de mecanismos de comunicación, dotar a los canales usados para conectar los componentes que componen dicha estructura de aplicación con un conjunto de características bien definidas, determinando el comportamiento de los componentes involucrados en cada una las conexiones establecidas. La elección de estas características de comunicación tiene un gran impacto en el rendimiento del sistema como se mostrará en la sección *Canales y Concurrencia* del capítulo *Ejemplos de Uso y Resultados*. Este trabajo se realizó colaborando con H. Posadas y A. Nicolás.

Formalización de la Comunicación

Para poder ser abstraídas como señales ForSyDe, lo elementos de comunicación no deben provocar ningún cambio en la secuencia de datos. Por tanto, cualquier cambio de orden, eliminación o inserción de datos, o cualquier cambio en los valores de los datos se han de considerar semánticamente compleja y, por tanto, ese "communication media" no podrá ser abstraído como una señal.

En este caso, estos "communication media" y los canales requerirán para su abstracción de un proceso ForSyDe para representar la funcionalidad que transforma los datos iniciales en una nueva secuencia de datos. Este proceso adicional proporciona el soporte formal para soportar comunicaciones complejas.

Modelado UML/MARTE de la Concurrencia y de canales

La concurrencia y la semántica de comunicación se localizan en tres elementos en el modelo UML/MARTE: se deriva de la combinación de propiedades de los componentes, interfaces y de los canales.

Concurrencia en los componentes de aplicación

Respecto a la estructura concurrente, el componente de aplicación puede ser caracterizado por un conjunto de hilos disponibles que serán usados para ejecutar llamadas a servicios que el componente proporcione. Este dato es capturado en el atributo "srPoolSize". El valor de este atributo define cuántos hilos concurrentes pueden ser creados dinámicamente por el componente, teniendo impacto en potenciales bloqueos.

Canales

La solución propuesta para especificar la semántica de canales se basa en una comunicación cliente/servidor, donde los componentes (que actúan como clientes) interactúan llamando a servicios de otros componentes (actuando como servidores). Los componentes pueden actuar como clientes y servidores al mismo tiempo, dependiendo de los servicios que requieren/proveen.

La semántica de los canales viene definida por la combinación de tres características principales: el tipo de dato transmitido, de la concurrencia y por otros detalles tales como las propiedades temporales. Las llamadas bloqueantes/no-bloqueantes, tanto al principio como al final de los servicios, junto con prioridades y otras restricciones definen la concurrencia resultante desde los canales. Además del tiempo, los canales pueden tener asociados búferes de almacenamiento, permitiendo estructuras del tipo "pipeline" y, además, habilita la partición de datos, lo que permite dividir los datos generando varias llamadas simultáneas.

De esta manera, flujos que están en principio totalmente desacoplados, pueden, en un momento dado, o confluir en un componente para sincronizar dichos flujos o porque dicho componente requiere de datos producidos en esos flujos desacoplados para poder implementar su funcionalidad. Así, el diseñador tiene un conjunto de elementos expresivos que puede usar de manera ágil y versátil para capturar diferentes propiedades, en relación con la concurrencia y la comunicación, en su aplicación y estudiar el impacto de dichas propiedades en el rendimiento de su sistema (sección *Canales y Concurrencia* del capítulo *Ejemplos de Uso y Resultados*).

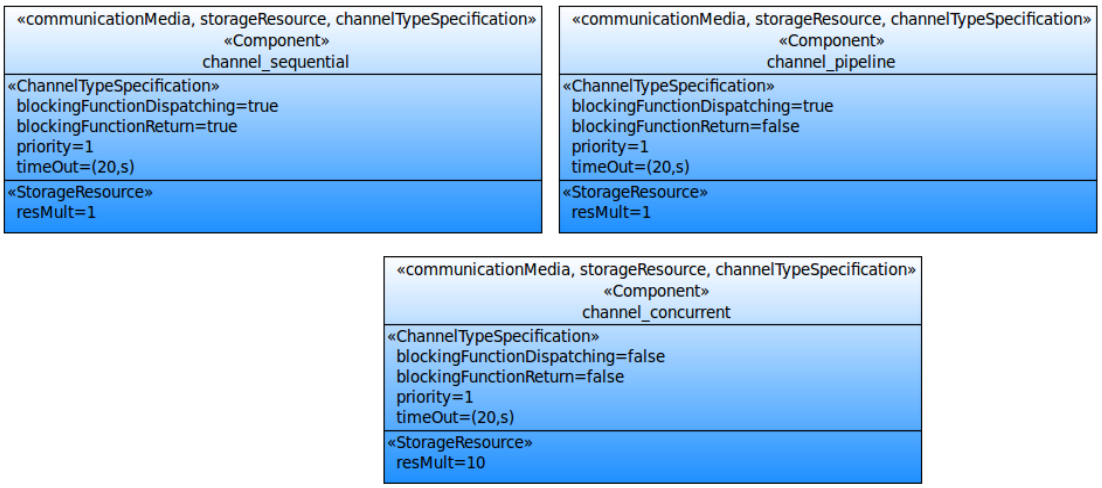


Figura 22 Ejemplos de tipos de canales

La Figura 22 muestra una serie de tipos de canales, lo cuales, según los valores de sus atributos, capturan una semántica de comunicación específica (secuencial, de pipeline o puramente concurrente).

El primer canal de la Figura 22 es usado para definir flujos de ejecución secuenciales (asociado a las propiedades bloqueantes); el segundo canal especifica una ejecución en pipeline ya que el cliente de la comunicación no necesita a esperar a que se completa la llamada al servicio; el último canal es usado para el caso general de flujos de ejecución concurrentes, ya que hay un búfer donde las peticiones entrantes pueden ser almacenadas hasta que son retiradas para ser ejecutadas y no tiene asociada ninguna propiedad bloqueante.

Interfaces

Los interfaces, además de definir los servicios proporcionados y requeridos, asegurando la compatibilidad entre los componentes para realizar las conexiones requeridas, tienen asociadas unas características semánticas que tienen impacto en la concurrencia. Esta semántica viene definida por la semántica de los servicios que se agrupan en dicho interfaz. Estos pueden ser:

- Secuencial: solo una invocación a un servicio puede darse a la vez.
- Protegida: múltiples invocaciones a un servicio pueden tener lugar, pero únicamente una puede comenzar.
- Concurrente: múltiples invocaciones a un servicio pueden tener lugar y pueden ejecutarse todas ellas a la vez.

Como regla de modelado, cada interfaz solo puede incluir servicios que tenga un solo tipo de las anteriores semánticas.

Tipos y tamaño de los datos

Los tipos de datos pueden ser definidos en el modelo para especificar los argumentos que serán transmitidos en las llamadas de los servicios de interfaz. Sin embargo, cuando el cliente es capaz de enviar más datos que los requeridos por el servidor, el modelado (y la herramienta de síntesis SW), permiten, de forma automática, la partición de los datos del cliente para lanzar varios servicios en paralelo, cada uno ejecutando con una parte de ese dato.

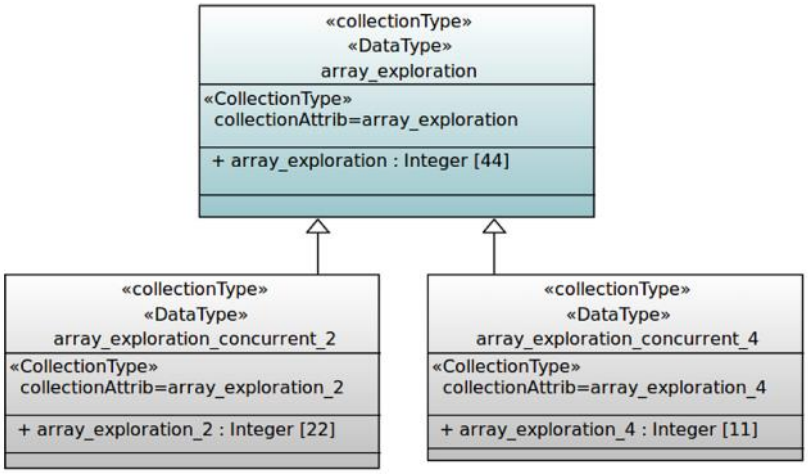


Figura 23 Ejemplo de tipos de datos para partición de datos

En la metodología que se presenta en esta tesis, solo se soportan para esta partición de datos los argumentos que son especificados como arrays de datos. La partición de datos viene definida por la diferencia de los tamaños del array de datos en argumentos de servicios de interfaz. La diferencia en los tamaños de los datos implica la creación de flujos concurrentes para computar cada parte de los datos, determinando el número de servicios que son ejecutados en paralelo. Por ejemplo, si el cliente llama a un servicio que tiene como argumento un array de 22 enteros, y el servidor usa ráfagas de 11 enteros, es posible lanzar dos servicios en paralelo, ejecutando cada uno de ellos 11 datos. Entonces, el cliente de la comunicación espera a que todos los servicios terminen, para juntar lo resultados obtenidos, antes de continuar.

Para soportar la partición de datos se requiere de una relación explícita entre los array de datos del cliente y del servidor, para asegurar la compatibilidad de los interfaces. Para modelar esto, se establecen relaciones entre los elementos del modelo que capturan este tipo de arrays, como se muestra en la Figura 23.

En esta figura, el tipo de dato “array_exploration” es modelado como una array de enteros de tamaño 44. Este tipo de dato se relaciona con otros dos: “array_exploration_concurrent_2” y “array_exploration_concurrent_4”, con un tamaño del array de 22 y 11, respectivamente. En el lado del cliente, el parámetro del servicio llamado debe ser especificado por el tipo de dato “array_exploration”, ya que es el tipo de dato real a transmitir en la comunicación. La relación entre el tamaño de los tipos de datos establece la estructura concurrente a generarse en el lado del servidor.

Los interfaces del cliente y del servidor para la partición de datos son capturados en el modelo usando herencia con un interfaz padre, para asegurar la compabilidad entre interfaces; esto es mostrado en la Figura 24. Esas nuevas interfaces (en la Figura 24, las interfaces "Interface_MEMC_TCTU_Exploration_2" y "Interface_MEMC_TCTU_Exploration_4") difieren del interfaz padre (Interface_MEMC_TCTU) en un parámetro, el cual cual es especificado por los tipos de datos mostrados en la Figura 23, respectivamente, habilitando la partición de datos.

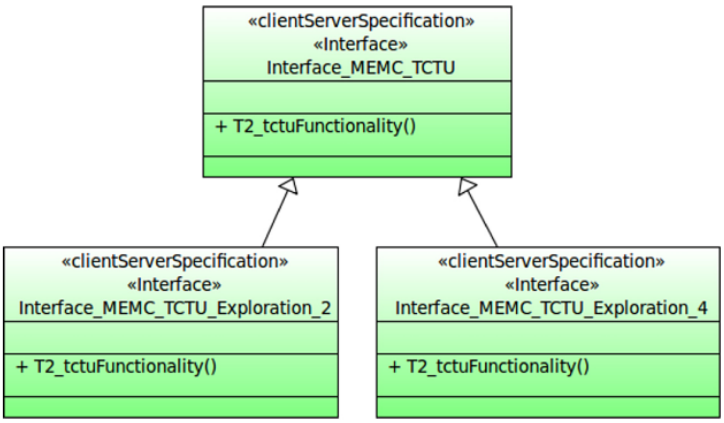


Figura 24 Herencia de interfaces para la exploración de la concurrencia

Unión de servicios

También se puede unir flujos de ejecución paralelos de otra forma diferente a la de partición de datos. A veces, un servicio requiere de dos clientes diferentes para ser ejecutado. Este problema puede ser resuelto usando los elementos de modelado descrito en la sección anterior pero invirtiendo los papeles de cliente y servidor. Por ejemplo, si un servicio C requiere llamar (y transferir datos de entrada) desde el servicio A y B para ser ejecutado, es posible cambiar el flujo de llamadas original de A → C (A llama C) y B → C, A→ C, entonces C → B y C continua. Como resultado, B ya no es un cliente, sino un servidor. Sin embargo, si A requiere de B datos devueltos por C para continuar su ejecución, la transformación puede necesitar inmanejables modificaciones en el modelo y en el código funcional.

Para abordar esta situación, una alternativa consiste en proporcionar la unión entre A y B como un canal de tipo "rendezvous", antes de ejecutar C. En este caso, se usan diferentes interfaces de comunicación en los lados del cliente y del servidor del canal. Todos esos interfaces tiene el mismo servicio asociado. Sin embargo, la declaración de dicho servicio es totalmente diferente en cada interfaz. Siguiendo el ejemplo anterior, en C el servicio se especifica completamente, caracterizando completamente los parámetros

de dicho servicio. Por otra parte, los interfaces A y B solo especifican parte de los parámetros del servicio. Por ejemplo, si A proporciona un buffer y B proporciona otro buffer y C requiere de ambos búferes, los interfaces han de incluir los correspondientes servicios como "func (bufferA)", "func (bufferB)" y "func (bufferA, bufferB)", respectivamente. Para capturar esto, lo interfaces A y B han de estar relacionados con el interfaz padre, C. Esto se muestra en la Figura 25, donde para realizar la función "stereo_matching", se necesitan dos imágenes (ver ejemplo de la sección *Canales y Concurrencia* del capítulo *Ejemplos de Uso y Resultados*).

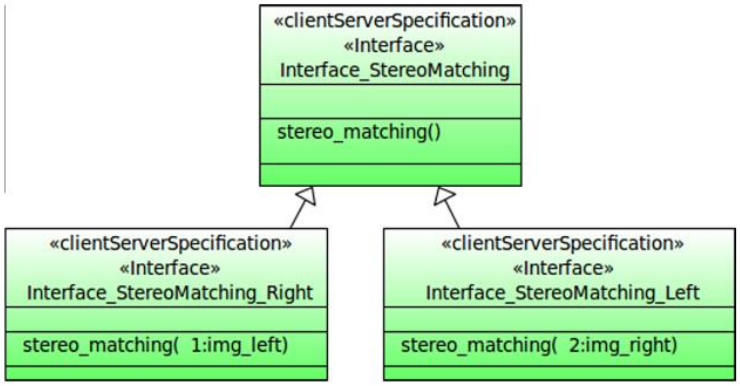


Figura 25 Herencia de interfaces para la unión de flujos concurrentes

En el trabajo [f)] (y en el manual [oo)]), se presenta la metodología de modelado que permite la captura de canales con diferentes propiedades que se han presentado en esta sección. Con la metodología descrita en dicho trabajo [f)], el diseñador tienen la capacidad de definir diferentes características en las interfaces, canales y componentes de aplicación que le posibilitan el diseño de un gran gama de potenciales implementaciones del sistema en lo que a estructura de concurrencia y semántica de comunicaciones se refiere, cada una de ellas con características funcionales específicas y con rendimientos diferentes. Ejemplos de esto son FIFOS, "rendezvous", llamadas síncronos y asíncronos, accesos protegidas, etc.

Acorde con los anteriores mecanismos de comunicación, diferentes estructuras concurrentes pueden ser inferidas; canales con una semántica bien definida, propiedades de los componentes e herencia de interfaces, que implican la creación dinámica de hilos concurrentes. Esto se presentó en [f)], donde se utilizó un codificador H264 para poder aplicar esta metodología y estudiar el impacto de diferentes estructuras concurrentes en el rendimiento del sistema. Este ejemplo se mostrará en la sección *Canales y Concurrencia* del capítulo *Ejemplos de Uso y Resultados*.

Proyectos en los que este trabajo se desarrolló:

1. PHARAON, [g)] y [aaa)]. Director principal: E. Villar; presupuesto: 354000€; periodo: 2011-2014.

Publicaciones resultantes:

1. Automatic synthesis of communication and concurrency for exploring component-based system implementations considering UML channel semantics, [f)].

2. Automatic synthesis from UML/MARTE models using channel semantics, [ii)].

3. Automatic Concurrency generation through Communication Data Splitting based on UML-MARTE Models, [y)].

4. Especificación del sistema: plataforma y heterogeneidad

Como se mencionó en la introducción, existe una gran variedad en lo que a recursos de plataforma HW/SW se refiere. Ese hecho abre un gran abanico de variables a considerar a la hora de abordar el diseño del sistema:

- El mapeo arquitectural para cada componente de aplicación.
- Tipos de recursos de procesamiento presentes en la plataforma.
- El sistema operativo.
- La infraestructura de comunicaciones.

En este contexto, primero se describen los mecanismos desarrollados para abordar la heterogeneidad desde el punto de vista de los recursos HW y el mapeado a dichos recursos de los componentes de aplicación en diferentes escenarios. De la misma forma, posteriormente se presentan distintos escenarios donde diferentes recursos de la plataforma SW son usados para definir la infraestructura de comunicaciones y ver el impacto de los mismos en el rendimiento del sistema.

4.1 Heterogeneidad en la plataforma HW

Respecto a los recursos HW, plataformas con varios procesadores, tipo de procesador, co-procesadores, DPS, GPUs son ejemplos de la diversidad recursos de cómputo que se pueden encontrar actualmente en dichas plataformas HW. Además de esto, caches, buses, memorias, componentes I/O, FPGAs,…hace que la paleta de recursos HW disponibles para poder modelar una plataforma sea muy amplia (Figura 26).

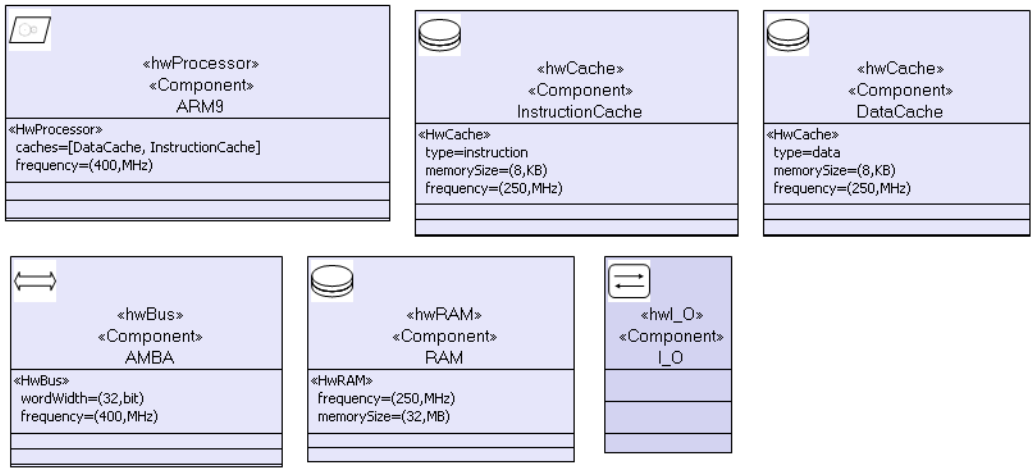


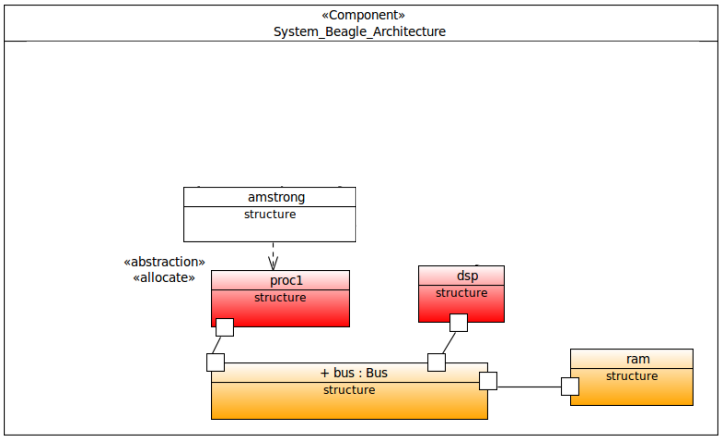
Figura 26 Diferentes Recursos HW

Con estos elementos, es posible modelar plataformas muy diferentes en lo que a su estructura y tipos de componentes se refiere. La Figura 27 muestra dos ejemplos de modelado de dos tipos de plataformas: la Figura 27A) muestra el modelo de una Beagle con un procesador (ARM Cortex-A8) y un DSP. La Figura 27B) muestra el modelo de una i.MX6, con cuatro procesadores (ARM Cortex-A9) y una GPU.

También dichos modelos incluyen la especificación del sistema operativo (Amstrong y Linux genérico, respectivamente). Una vez que la plataforma está definida, lo siguiente a realizar es el mapeo arquitectural de la aplicación.

En el manual [oo)] está toda la información referente a los diferentes componentes HW y los atributos de los mismos que se usan en esta metodología.

A)



B)

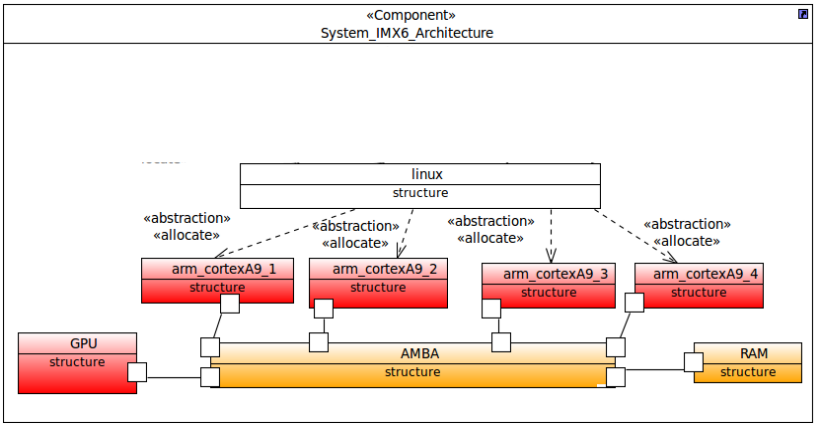


Figura 27 Diferentes Plataformas HW

4.2 Heterogeneidad en la plataforma SW

La plataforma SW también presenta una gran variedad en sus recursos a utilizar; numerosas APIs pueden ser utilizadas para realizar la implementación SW de la aplicación sobre dichas plataformas.

En el trabajo [v)] se muestra como diferentes APIs se pueden seleccionar en el modelo para poder realizar la implementación de los canales de comunicación así como de la estructura concurrente capturada en el PIM. La API considerada por defecto es POSIX; en el modelo se puede seleccionar cuál de los posibles mecanismos de comunicación que POSIX proporciona se selecciona (FIFOs, colas de mensajes, memoria compartida, archivos y sockets). Además de POSIX, se pueden capturar otras APIS en modelo:

- OpenMP: un estándar para generación de tareas concurrentes y su sincronización.
- OpenStream: que es una extensión de OpenMP orientada a explotar el paralelismo en entornos de memoria compartida para flujos de datos.
- MCAPI está orientada a síntesis de SW que permite implementar la concurrencia, sincronización y comunicación de dicho sistema concurrente.
- TCP/IP: protocolo de transmisión de datos.

Además de esto, el modelo puede incluir los controladores para ser instalados y poder hacer uso de un recurso de plataforma. En la Figura 28 se muestra la especificación de un controlador para poder manejar el DSP de la plataforma Beagle (Figura 27 A)).

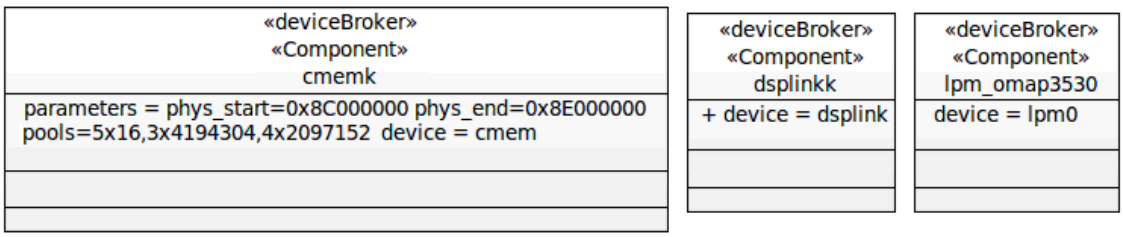


Figura 28 Controlador para manejar un DSP

En el manual [oo)] y [rr)] está toda la información referente a los diferentes recursos SW considerados en esta metodología.

Proyectos en los que este trabajo se desarrolló:

1. *COMPLEX, [32], [xx)]. Director principal: E. Villar; presupuesto: 325588€; periodo: 2009-2013.*
2. *PHARAON, [g)] y [aaa)]. Director principal: E. Villar; presupuesto: 354000€; periodo: 2011-2014.*
3. *CONTREX, [ww)], [h)], [ll)]. Director principal: E. Villar; presupuesto: 391336; periodo: 2013-2016.*
4. *DREAMS, [zz)]. Director principal: P. Sánchez; presupuesto: 141086€; periodo: 2011-2013.*

Publicaciones resultantes:

1. *Automatic Synthesis over multiple APIs from UML/MARTE Models for easy Platform Mapping and Reuse, [v)].*
2. *Automatic Deployment of Component-Based Embedded Systems from UML/MARTE Models Using MCAPI, [x)].*
3. *Automatic synthesis of Embedded SW Communications from UML/MARTE models supporting memory-space separation, [dd)].*

5. *Especificación del sistema: Mapeo Arquitectural*

Un requerimiento de diseño de los sistema empotrados es la asignación de la aplicación en una plataforma con recursos HW/SW. Para ello, la primera posibilidad es asignar cada componente de aplicación del sistema a un recurso HW.

En esta línea, cada componente de aplicación puede ser asociado a recursos HW y mapeado de manera automática, ya que la funcionalidad asociada a los componentes debe ser independiente de plataforma.

Para generar el código de aplicación que será utilizado en las diversas etapas del proceso de diseño, como análisis de modelos virtuales o prototipos, lo primero es asociar a cada componente de aplicación el conjunto de archivos y carpetas que son necesarios para su compilación, Figura 29.

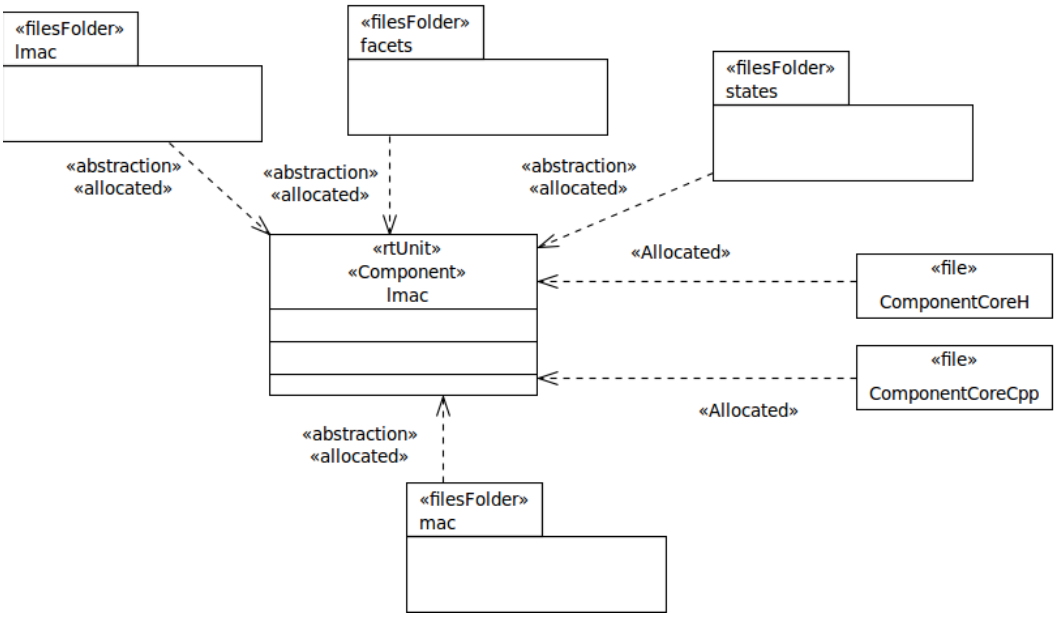


Figura 29 Asociación de archivos y carpetas a un componente de aplicación

Como se dijo en la sección *Modelo Independiente de Plataforma*, la funcionalidad del elemento de cómputo puede ser modelada en esta metodología mediante archivos, donde dicha funcionalidad ha sido implementada. El código de esta funcionalidad también ha de tener una serie de propiedades. En concreto, esté código a ser independiente de plataforma, sin ninguna llamada a cualquier servicio de sistema operativo, con el fin de poder realizar la síntesis de SW para cualquier tipo de plataforma y a cualquier tipo de recurso que esté presente en ella.

También, para realizar la generación de los ejecutables, es necesaria la definición de las librerías auxiliares, asociando dichas librerías al componente sistemas de la aplicación, como puede verse en la Figura 30.

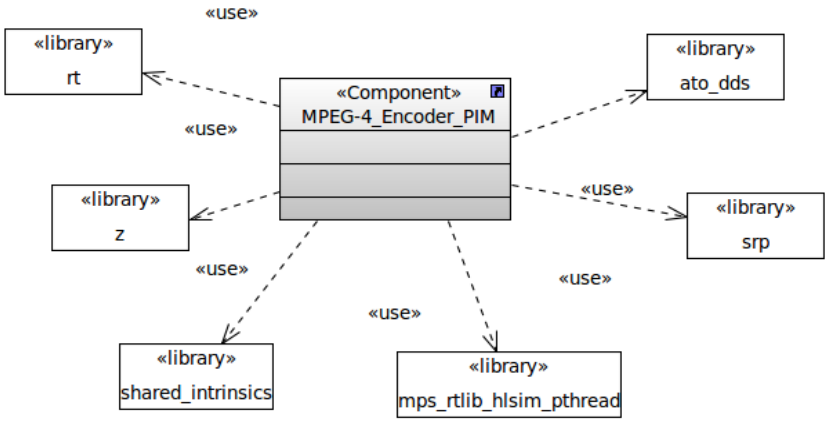


Figura 30 Asociación de librerías para la compilación

Con toda esta información (descrita en más detalle en [oo)] y en [rr)]) es posible además realizar el proceso de compilación. En este proceso hay que distinguir dos escenarios de compilación: una nativa, a partir de lenguaje de implementación de los componentes de aplicación se usa el compilador de forma automática (gcc o g++ según el lenguaje). En el segundo escenario, de compilación cruzada, es necesario tener detalles relacionados con la plataforma destino y los recursos HW que ella tiene; esto será descrito en las siguientes secciones.

Además de esto, la metodología permite el refinamiento de la funcionalidad de los componentes de aplicación. Como se mostró en la sección *Modelo Independiente de Plataforma*, para permitir la síntesis, los componentes de aplicación tienen asociado el conjunto de archivos que representan el código de implementación de dicho componente. La Figura 31 muestra el refinamiento de archivos de implementación que, partiendo de una implementación genérica, se refinan con código optimizado para un cierto recurso de plataforma. En este caso, estos recursos son un DSP y el co-procesador NEON. Este ejemplo se mostrará con más detalle en la sección *Exploración de Recursos HW* del capítulo *Ejemplos de Uso y Resultados*.

No obstante, el tratamiento de cada componente de aplicación por separado puede dar lugar a ineficiencias innecesarias. Para resolver este problema la metodología desarrollada permite que varios componentes pueden ser agrupados con el fin de

mapearlos en un mismo recurso HW por razones de utilización del mismo, uso exclusivo del recurso para una función específica,…

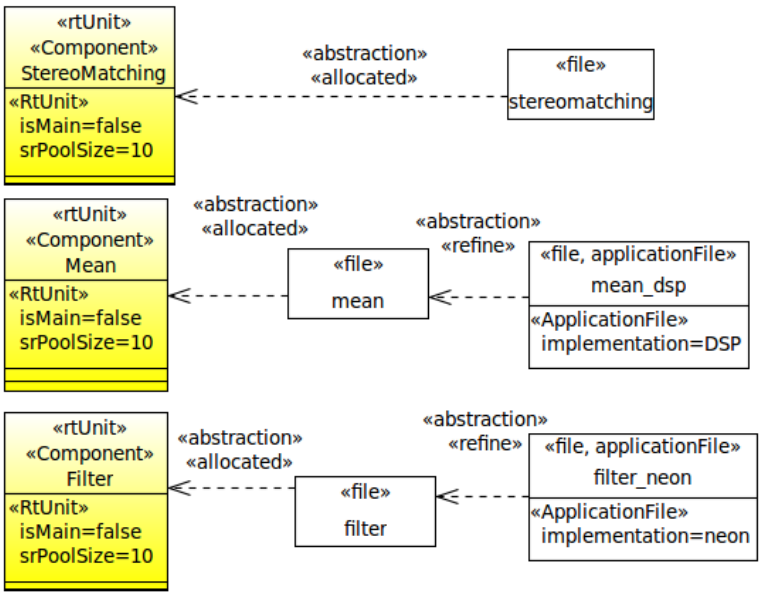


Figura 31 Refinamiento de archivos para el mapeo a recursos HW.

Algunos componentes pueden tener dependencias específicas que limitan las posibilidades de mapeo en la plataforma. Por ejemplo, el código funcional de dos componentes del sistema puede depender de una variable compartida común. Por lo tanto, requiere que dichos componentes se asignen juntos para el correcto funcionamiento de la aplicación, ya que, en caso contrario, cada componente tendría una copia diferente de dicha variable, pudiendo aparecer riesgos de incoherencias en los datos.

Además, podemos encontrar que el código asociado a dos componentes tiene variables globales o funciones con los mismos nombres, obligándolos a ser compilados y enlazados por separado. Para resolver todos estos problemas la metodología presentada en este trabajo permite la definición de espacios de memoria donde los componentes de aplicación son agrupados.

Además de todo lo anterior, existen otras ventajas por el hecho de considerar los espacios de memoria en el proceso de diseño. En el mercado actual de plataformas para sistemas empotrados existe una gran variedad de recursos. Esta heterogeneidad viene dada por la diferente naturaleza de los recursos de procesamiento que pueden estar presentes en dichas plataformas como, por ejemplo, DSPs, GPUs, procesadores y coprocesadores (NEON), HW específico. A esto hay que añadirle los recursos SW de la plataforma, APIs… De esta forma, con los espacios de memoria se puede agrupar los

componentes que se desee para ser mapeados a un recurso HW de cómputo específico, o para que se use una API concreta para su implementación.

Los espacios de memoria y la agrupación de los componentes de aplicación en los mismos se definen en la "MemorySpaceView".

En una primera fase del desarrollo de la metodología (asociada al trabajo [b)]), la noción de espacio de memoria no estaba presente. Sin embargo, este hecho presentaba múltiples limitaciones en lo que a analizar el sistema. Como se ha mencionado con anterioridad, en las plataformas actuales no solo hay procesadores como elementos de cómputo, también hay otros dispositivos como co-procesadores o GPP-GPUs que pueden ser usados como recursos ejecutivos. Además de esto, la amplia variedad de recursos SW disponibles usados para la implementación de las comunicaciones de los diferentes elementos de la aplicación hace que el abanico de potenciales aspectos a considerar por el diseñador sea muy amplia.

En este contexto, la metodología de modelado se enriqueció con más elementos expresivos que permitieran capturar esa heterogeneidad de recursos de la plataforma y, de esa forma, dotar al diseñador de más herramientas en su proceso de desarrollo. El hecho de introducir los espacios de memoria dota de más versatilidad al diseñador desde el punto de vista funcional y de mapeo, al poder considerar plataformas con otros tipos recurso de procesamiento que no solo sean procesadores. Por tanto, la consideración de los espacios de memoria proporciona dos ventajas muy relevantes: agrupar funcionalidad y permitir el mapeo de dicha funcionalidad en plataformas donde la variedad de recursos HW y SW está muy presente.

Este concepto se presentó en el trabajo [c)] donde se presenta la introducción de los espacios de memoria y cómo son usados para hacer el mapeo en la plataforma.

Por último, la tarea del mapeo arquitectural del diseño implica la creación de toda la infraestructura necesaria que permita la ejecución de la aplicación en dicha plataforma. Esta infraestructura se compone, entre otras cosas, de los canales de comunicación necesarios para conectar los diferentes componentes de la aplicación. Estos canales de comunicación dependerán de las propiedades de los interfaces y de los canales usados para conectar los componentes de aplicación, como se mostró en la sección anterior *Canales y Concurrencia*. Su implementación se describirá en la siguiente sección.

Proyectos en los que este trabajo se desarrolló:

1. PHARAON, [g)] y [aaa)]. Director principal: E. Villar; presupuesto: 354000€; periodo: 2011-2014.

Publicaciones resultantes:

1. Automatic synthesis of embedded SW for evaluating physical implementation alternatives from UML/MARTE models supporting memory space separation, [c)].

2. Automatic Synthesis over multiple APIs from UML/MARTE Models for easy Platform Mapping and Reuse, [v)].

3. Automatic synthesis of Embedded SW Communications from UML/MARTE models supporting memory-space separation, [dd)].

6. Información asociada a la síntesis

Como se describió en la sección anterior, la funcionalidad independiente de plataforma es asociada al modelo en función de archivos fuente o librerías. Sin embargo, para poder generar el código ejecutable final es necesario crear otros códigos que relacionen la funcionalidad independiente de plataforma con la plataforma destino. En este trabajo se han incluido dos secciones sobre aspectos relacionados con la síntesis.

6.1 Recursos HW

Un caso en el que la metodología presentada en esta tesis cubre un proceso de síntesis de SW es la generación de toda la infraestructura necesaria para poder ejecutar la aplicación en una determinada plataforma y sobre un recurso específico.

Como fue descrito en la sección *Especificación del sistema: Mapeo Arquitectural*, los componentes de aplicación son mapeados en espacios de memoria para agrupar funcionalidad y poder realizar un mapeo de dicha funcionalidad a un recurso HW específico de la plataforma. Con el mapeo realizado, todo el entorno de desarrollo implementado permite obtener, como resultado de su aplicación, un ejecutable compilado específico asociado al correspondiente recurso HW al cual el espacio de memoria ha sido mapeado.

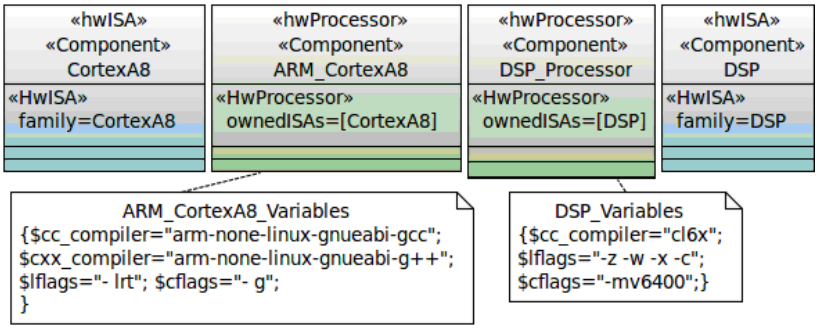


Figura 32 Recursos HW y sus compiladores y opciones de compilación asociados

Para permitir esto, el modelo no solo debe capturar el tipo de recurso de procesamiento sino toda la información necesaria para su utilización. Información sobre compiladores, opciones de compilación y de enlazado, son añadidos al modelo para poder ejecutar código sobre estos recursos HW. En la Figura 32 se muestran dos ejemplos de cómo un componente que representa un procesador del tipo ARM CortexA8 se le asocia los compiladores para C y C++, así como las correspondientes opciones de compilación y enlazado.

Otro ejemplo de tipo de recurso HW sobre el que se hará síntesis de código será GPU, que se mostrará en la sección *Exploración de Recursos HW* del capítulo *Ejemplos de Uso y Resultados*.

Más detalles en de la síntesis sobre recursos HW en [rr)].

6.2 Recursos SW

También es posible decidir qué tipo de interfaz de comunicaciones (y su librería asociada) ha de ser usada para la implementación de la comunicación entre dos componentes de aplicación, así como de la estructura concurrente de dichos componentes. Las interfaces (APIs) soportadas fueron mostradas en la sección *Heterogeneidad en la plataforma SW* de este capítulo.

En el caso de que no se seleccione ninguna de estas interfaces, también es posible seleccionar diferentes mecanismos de implementación de las comunicaciones ofrecidos por un sistema operativo. Estos pueden ser canales FIFO, "sockets", cola de mensajes, memoria compartida y archivos.

Como librería por defecto para implementar la estructura concurrente es POSIX.

De esta forma, el diseñador puede seleccionar la API que mejor se adapte a sus requerimientos. En la sección *Exploración de Recursos SW* del capítulo *Ejemplos de Uso y Resultados* se incluye una batería de ejemplos donde se muestra el impacto de los diferentes recursos SW para implementar las comunicaciones y la estructura concurrente de la aplicación.

En el manual [oo)] y [rr)] está toda la información referente a los diferentes recursos SW considerados en esta metodología.

Proyectos en los que este trabajo se desarrolló:

1. *PHARAON, [g)] y [aaa)]. Director principal: E. Villar; presupuesto: 354000€; periodo: 2011-2014.*

Publicaciones resultantes:

1. *Automatic synthesis of communication and concurrency for exploring component-based system implementations considering UML channel semantics, [f)].*
2. *Automatic synthesis of embedded SW for evaluating physical implementation alternatives from UML/MARTE models supporting memory space separation, [c)].*
3. *Automatic Synthesis over multiple APIs from UML/MARTE Models for easy Platform Mapping and Reuse, [v)].*
4. *Automatic Deployment of Component-Based Embedded Systems from UML/MARTE Models Using MCAPI, [x)].*
5. *Code Synthesis of UML/MARTE models for physical platforms considering resource-specific codes, [aa)].*
6. *System synthesis from UML/MARTE models: The PHARAON approach, [cc)].*
7. *Automatic synthesis of Embedded SW Communications from UML/MARTE models supporting memory-space separation, [dd)].*

6.3 Sistemas OpenMAX

Un caso particular es la metodología son los sistemas OpenMAX. El estándar OpenMAX [17] es una iniciativa promovida por el Grupo Khronos que define una interfaz estandarizada de interfaces de componentes para permitir la integración "códecs" multimedia implementados en hardware o software. Esto surgió de la colaboración con el departamento ARCO de la Universidad de Castilla-La Mancha.

En este contexto se enriqueció la metodología de modelado con nuevos aspectos para poder abordar en modelado de este tipo de sistemas y así poder realizar una posterior síntesis HW (sección *OpenMAX* del capítulo *Entorno de Desarrollo: Integración de Herramientas*)

Componentes de Aplicación

Los componentes de aplicación se enriquecieron con nuevas propiedades que poder capturar aquellos aspectos que caracterizan a los componentes OpenMAX.

Las tasas de datos (entrada o salida) indican la velocidad de datos para los flujos de datos entrantes o salientes que el componente admite. También se debe definir el tiempo de procesamiento de componente. Cada componente de OpenMAX tiene asociado un modo de comportamiento que determina cómo funciona. El modo del componente define la forma en que los datos son enviados por el componente:

- "full": los buffers del componente deben estar llenos para el envío de datos.
- "continuos": los datos son enviados en un flujo continuo, mientras los datos se van almacenan en los búferes.

El factor de compresión define la relación entre los flujos de datos de entrada y de salida. La longitud de ráfaga define el tamaño del flujo de datos utilizados por el componente. Por último, cada componente OpenMAX tiene una dirección de memoria con el fin de identificar al componente para poder realizar la transmisión de datos. Además, cada componente tiene asociado una memoria.

Todas estas propiedades son mostradas en la Figura 33. Cómo son capturadas estas propiedades en el modelo es explicado en más detalle en [d)].

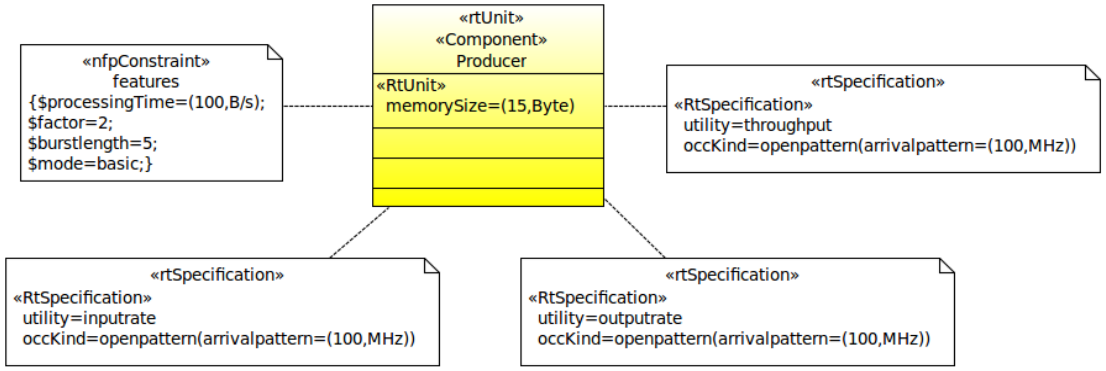


Figura 33 Modelado de un componente OpenMAX.

Todas las características anteriores se modelan a nivel de componente, lo que implica que múltiples instancias del mismo componente tienen las mismas características asociadas. Sin embargo, la dirección de memoria tiene que ser capturada a nivel de instancia, ya que dicha dirección tienen que ser unívoca para cada objeto.

Canales de Comunicación

Los componentes OpenMAX están interconectados a través de puertos y canales. Los puertos están definidos para modelar las comunicaciones orientadas a flujos de datos, especificando la latencia de dicho enlace.

Como canales de comunicación específicos para estos componentes se distinguen dos canales diferentes. El primer tipo canal viene caracterizado por medio de búferes (Figura 34). Estos búferes se asocian a los puertos de los componentes de aplicación. Un búfer puede ser local al componente de la aplicación o compartido con otros componentes de la aplicación. Para especificar este último tipo de búferes, se caracterizan por el elemento "SharedDataComResource" (Figura 34). Un mismo puerto puede tener un conjunto búferes con diferentes características.

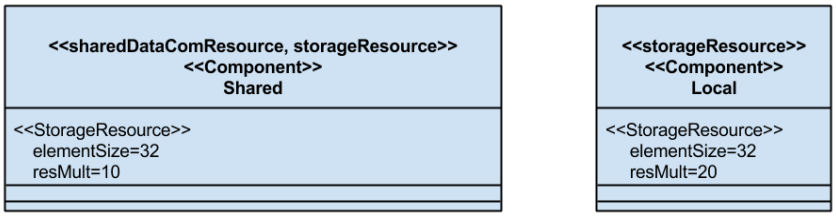


Figura 34 Modelado de los canales usados para comunicar componentes OpenMAX.

Hay una restricción de modelado: todos los búferes asociados a un mismo puerto deben tener el mismo tamaño; la cantidad de datos que pueden ser almacenados también tiene que ser el mismo para poder permitir la generación automática durante el proceso de síntesis.

Cómo son capturadas estas propiedades en el modelo es explicado en más detalle en [d)].

Proyectos en los que este trabajo se desarrolló:

1. *DREAMS, [zz)]. Director principal: P. Sánchez; presupuesto: 141086€; periodo: 2011-2013.*

Publicaciones resultantes:

1. *Synthesis of Simulation and Implementation Code for OPENMAX Multimedia Heterogeneous Systems from UML/MARTE models, [d)].*
2. *UML/MARTE methodology for automatic SystemC code generation of OPENMAX multimedia applications, [bb)].*

7. Sistemas Distribuidos

La metodología de UML/MARTE presentada en este trabajo también permite el modelado de sistemas distribuidos. Esta metodología de modelado llena el vacío en las metodologías de modelado UML/MARTE actualmente disponibles, adecuadas para el modelado de sistemas empotrados, pero ineficientes para el modelado de redes.

La metodología soporta la descripción de la arquitectura de red, basándose en nodos y canales. Los nodos y los canales permiten el modelado de alto nivel de recursos de red; sus atributos reflejan capacidad de cómputo y comunicación, respectivamente.

Esta metodología soporta un amplio concepto de nodo, un elemento fundamental en el modelado de redes. Estando orientado a sistemas distribuidos empotrados, la metodología cubre modelos donde el nodo es entendido como un elemento computacional con capacidad de interfaz de red. Además, la metodología apoya el modelado de nodos de cualquier tipo (switches, routers, centros de datos, superordenadores,…) que cubren diferentes capacidades computacionales y funcionales, a diferentes niveles de abstracción.

Además, la metodología soporta el modelado de la aplicación distribuida definiendo tareas y flujos de datos. Las tareas se asignan a los nodos. Los flujos de datos reflejan la comunicación tarea a tarea sin una coincidencia necesaria con un canal o ruta de comunicación fija.

Como ejemplos de sistemas distribuidos que cubre esta metodología se van a mencionar tres casos, cuando el sistema operativo está distribuido entre los nodos, cuando los nodos son entidades competas HW/SW y las redes de sensores

En el documento [pp)] se describe completamente la metodología, así como todos los aspectos que se pueden modelar de los sistemas distribuidos.

Este trabajo se realizó en base al trabajo previo realizado por E. Ebeid y D. Quaglia de la Universidad de Verona y presentados en [89], [90] y con la colaboración de F. Herrera y E. Villar.

7.1 Sistema Operativo Distribuido

La metodología permite la captura de sistemas operativos distribuidos. La metodología asume que tal mapeo se puede hacer para:

- Cualquier nodo que se describa sólo ha de tener de recursos HW para conformar su plataforma.
- Cualquier nodo con una descripción de arquitectura interna con recursos computacionales (CPUs) no asignados a un sistema operativo.

En ambos casos la semántica por defecto es que el sistema operativo controla a todos los recursos computacionales disponibles de los nodos a los que dicho sistema operativo está asociado (Figura 19 y Figura 35, realizada por F. Herrera). Los componentes de aplicación (o espacios de memoria) son mapeados al sistema operativo para su ejecución en los nodos de cómputo disponibles (Figura 35).

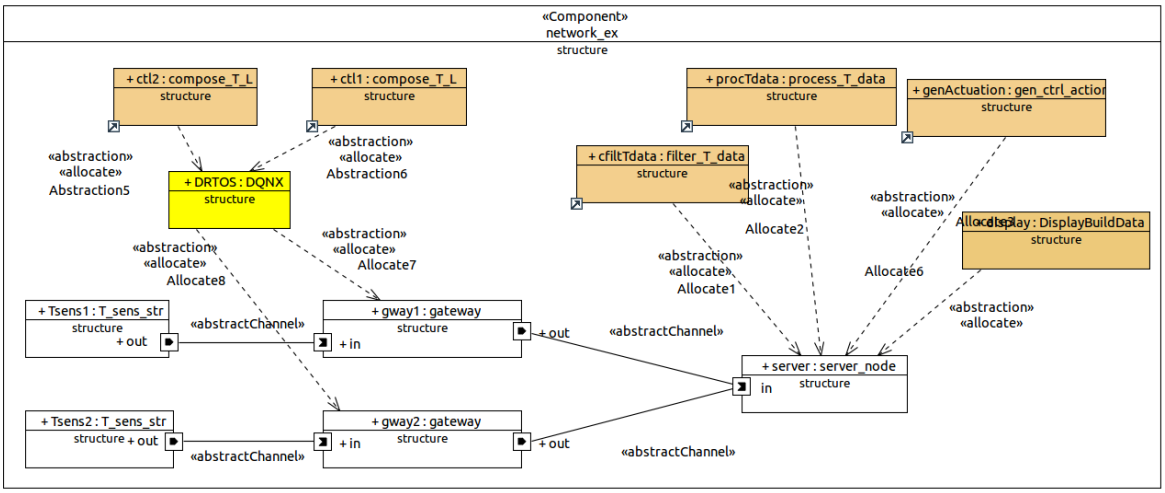


Figura 35 Sistema operativo distribuido.

7.2 Sistema de Nodos HW/SW

La otra situación típica es la de tener un conjunto de nodos caracterizados por tener una plataforma HW/SW, al estilo de los que muestran las figuras Figura 17 y Figura 18. Sin embargo, en este escenario, la aplicación es mapeada de la forma que la Figura 36 (realizada por F. Herrera) muestra.

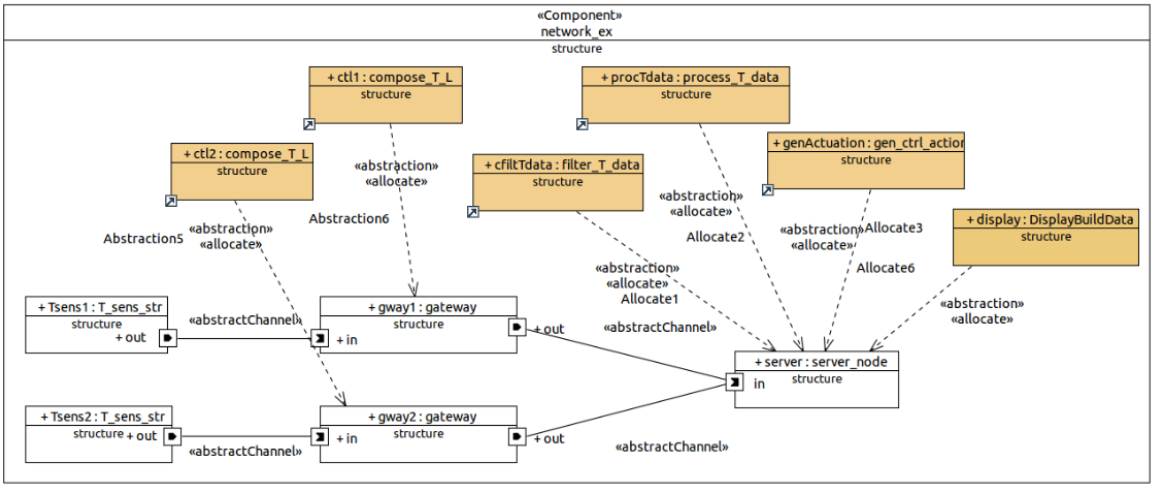


Figura 36 Sistema distribuido

Para más ejemplos de sistemas destruidos que la metodología cubre, ver el manual [pp)].

Proyectos en los que este trabajo se desarrolló:

1. CONTREX, [h)], [ll)] [ww)]. Director principal: E. Villar; presupuesto: 391336; periodo: 2013-2016.

7.3 Redes de sensores

Las redes de sensores (“Wireless Sensor Networks”, WSNs) son sistemas de gran complejidad por sí mismos, ya que se han de considerar un gran número de variables como dónde instalarlo, número de nodos de tu red, tipo de nodo… A esto hay que añadir que los nodos no están en un entorno inerte; éste afecta a la de la red de forma que puede provocar disfuncionalidades que reduzcan su rendimiento.

Por esto se requiere de entornos de trabajo que permitan al diseñador la especificación y simulación de la red, estableciendo un proceso de exploración y de refinamiento de la red, considerando todas las variables previas, con el fin de encontrar una configuración de la red que asegure su correcto funcionamiento.

Para este fin se desarrolló un entorno de diseño que permitiera, mediante modelos UML/MARTE, especificar y simular una red WSN para estudiar su rendimiento. Este

rendimiento viene definido por el consumo de potencia de cada nodo. Esto se debe a que una de las principales limitaciones en WSNs es normalmente sus requisitos de baja potencia; los nodos son normalmente dispositivos que funcionan con baterías, en los cuales su vida útil depende de la duración de dicha batería.

Este trabajo se realizó en colaboración con P. Sánchez y, principalmente, con A. Díaz y J. Medina.

Como se mencionó con anterioridad, estas redes pueden estar desplegadas en entornos hostiles que afecten a su funcionamiento. Por esta razón, este entorno de diseño ha de permitir la especificación y simulación de agentes externos a la red que interfieran con ella. Esto proveerá de información al diseñador para rediseñar su red con el fin de protegerla contra tales efectos dañinos. Los ataques son descritos en la sección *Ataques a Redes* de la sección *Modelado del Entorno* de este capítulo.

Una primera versión de este trabajo fue [r)], que abarcaba el modelado de la red y únicamente consideraba un solo tipo de atacantes. Sin embargo, este trabajo presentaba un conjunto de carencias; primero a la capacidad de proporcionar mecanismos para modelar diferentes aspectos de estos sistemas y también el cómo se capturaba dicha información en el modelo (los diferentes mecanismos expresivos de UML y MARTE usados). Tras analizar las desventajas de lo presentado en [r)], se cambió gran parte del enfoque metodológico. El resultado final de toda esta evolución es presentado en [a)]. En dicho trabajo ya es posible capturar un gran número de tipos de atacantes, así como especificar un gran número de las propiedades de la red y de cada uno de sus nodos, de tal forma que sea más fácil su modificación para hacer que el proceso de exploración fuese más rápido.

Este proceso de exploración se hará de forma semiautomática; este proceso se describirá en detalle en la sección *Exploración semiautomática* de capítulo *Entorno de Desarrollo: Integración de Herramientas*. Un ejemplo de esto se mostrará en la sección *Redes de Sensores* del capítulo *Ejemplos de Uso y Resultados*.

Interfaz de Red

Como se mostró en la sección *Vistas: Modelo Específico de Plataforma para Sistemas Distribuidos,* los nodos se pueden especificar su estructura interna, en base a una plataforma HW/SW y a la aplicación que ellos ejecutan.

En este contexto es preciso detenerse en uno de estos componentes que será relevante para caracterizar el nodo; este es el interfaz de red.

El modelado de las interfaces de red (el cual fue propuesto por E. Ebeid en los trabajos [89] y [90]) es de especial interés, ya que es un dispositivo que es crítico en la evaluación del rendimiento del sistema y en el modelado de los efectos de los ataques externos. En este contexto, el atributo "txPower" denota la potencia asociada a la comunicación por parte de nodo.

Caracterizando las comunicaciones

De la misma manera, se debe especificar las propiedades de las conexiones entre los nodos. Estas están caracterizadas por tres propiedades, modeladas como variables de diseño. Estas son "$transmPower", $numOfRetries y $encrypted. "$transmPower" indica la potencia de transmisión utilizada para el envío de un paquete en esa conexión nodo-nodo; $numOfRetries contiene el número de reintentos para la transmisión del mismo paquete; $encrypted indica que el paquete está cifrado o no para su transmisión. Estas variables de modelado tienen valores específicos dependiendo de la conexión establecida entre los nodos.

Estas variables, junto con el atributo "txPower", definen el consumo de energía del interfaz de red. Cuando se combinan con el consumo de los otros componentes HW del nodo (procesadores, buses, memorias…), esto permite analizar el consumo completo del nodo, pudiendo estudiar el ciclo de vida de la batería del nodo.

Finalmente, hay una propiedad final que influye en la transmisión de los paquetes; cada transmisión de datos tiene asociada una probabilidad de éxito. En la Figura 37 se denota esto por el atributo "prob". Esta probabilidad de éxito puede verse modificada por la acción de algún atacante del entorno (*Ataques a Redes* de la sección *Modelado del Entorno* de este capítulo).

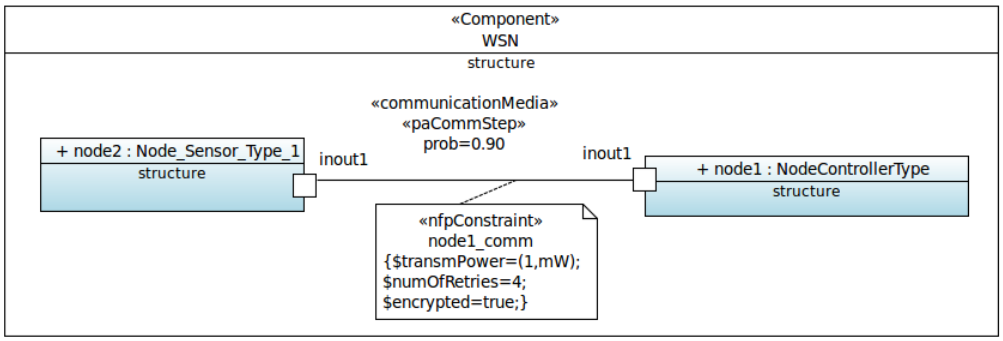


Figura 37 Especificación de las comunicaciones entre nodos

Para más detalles [a)] y [pp)].

Proyectos en los que este trabajo se desarrolló:

1. *TOISE, [yy)]. Director principal: P. Sánchez; presupuesto: 163861€; periodo: 2011-2013.*

Publicaciones Resultantes:

1. *High-Level design of wireless sensor networks form performance optimization under security hazards, [a)].*
2. *Modeling and Simulation of secure wireless sensor network, [r)].*

8. Modelado del Entorno

Durante todas las secciones anteriores se ha estado abordando el modelado del sistema cómo un modelo de alto nivel que permite el desarrollo de diferentes etapas, donde un aspecto del sistema es definido y, posteriormente, analizado. Sin embargo, no suele ser habitual que los sistemas sean entes autónomos que viven una vida aislada sin interactuar con nada. Normalmente, los sistemas están situados dentro de un entorno con el que se establece un conjunto de relaciones que se han de considerar durante el proceso de diseño.

Para desarrollar este modelado del entorno se han usado mecanismos del propio UML, de MARTE y de un tercer lenguaje llamado UTP ("UML Testing Profile") [30] (el uso de UTP fue una idea original de E. Villar). UTP es usado para especificar en el modelo los diferentes componentes que conforman el entorno que interactúa con el sistema. De esta forma, en el modelo, los componentes del sistema y del entorno están claramente diferenciados.

La metodología permite modelar el entorno dependiendo cómo se entienda la interacción del entorno con el sistema.

Hay otro escenario de modelado del entorno de un sistema. En este caso, se modela el entorno de las redes de sensores como elementos atacantes que interfieren con el correcto funcionamiento de la red.

8.1 Modelo de Estímulos

Un primer entorno considerado es el entorno como un generador de estímulos. Se definen diferentes escenarios de relación entorno-sistema según los diferentes casos de uso que se han de considerar durante el proceso de diseño. En estos escenarios se captura cómo se comunican el entorno y el sistema según una secuencia de llamadas a servicios que denotan cómo se sincronizan y qué información se transmiten entre ellos.

Toda la información referente a la estructura del entorno se ha de capturar en la vista de verificación.

Lo primero a capturar son los componentes que van a conformar el entorno. En la Figura 38 se muestra en ejemplo de entorno que se usará para el caso de uso presentado en la sección *Exploración del Espacio de Diseño* del capítulo *Ejemplos de Uso y Resultados*.

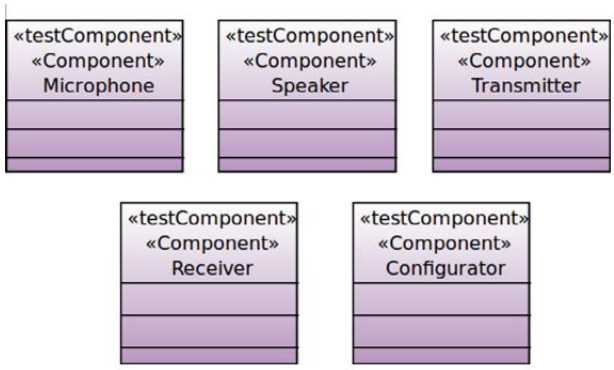


Figura 38 Componentes de entorno

Los componentes destacados en ella como "TestComponent" son los elementos del entorno:

- "Microphone": proporciona un número fijo de tramas al sistema de acuerdo a un tipo de test.
- "Transmitter": recibe las tramas que han sido codificadas desde las entradas envidas por el "Microphone".
- "Receiver": este componente emula la recepción de las tramas codificadas y entregas a al sistema para su decodificación.
- "Speaker": recibe la salida del decodificador.
- "Configurator": establece el modo de funcionamiento del sistema; o en modo continuo o discontinuo. En el último caso, la funcionalidad de detección de voz está habilitada, y el codificador puede hacer una mejor compresión si no hay voz, solo si se detecta ruido.

Con estos componentes se crea la estructura del entorno, como se muestra en la Figura 39.

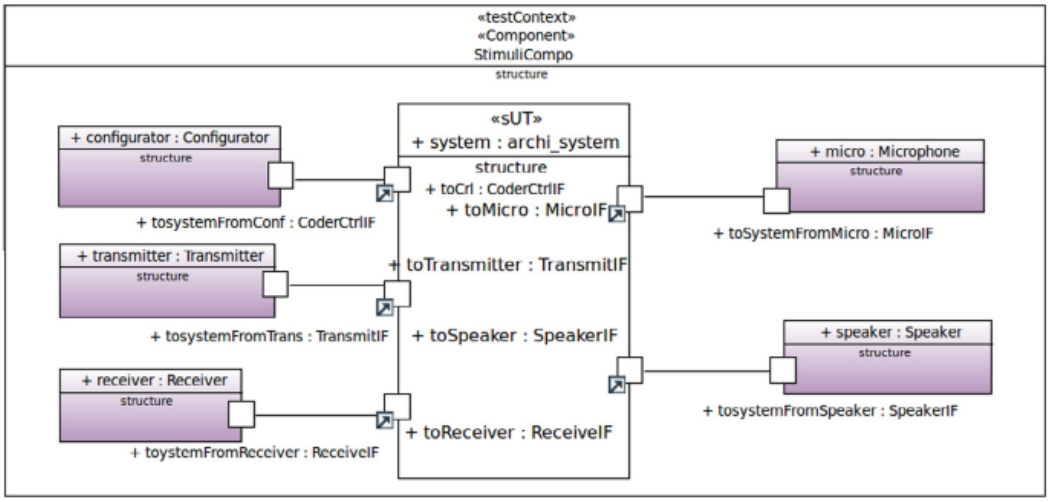


Figura 39 Estructura del entorno

Después de esto se crean los escenarios donde se van a definir las secuencias de estímulos que van a interactuar con el sistema (Figura 40).

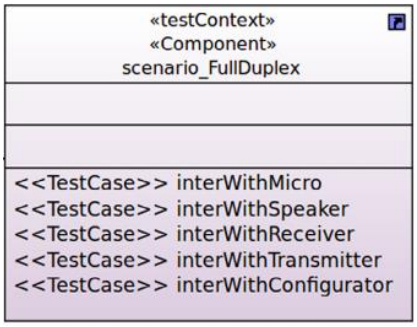


Figura 40 conjunto de escenarios

Los escenarios de estímulos se capturan mediante diagramas de secuencia. En ellos se modela la interacción entre el sistema y el entorno por media de secuencias de llamadas a servicios de interfaz. Estas llamadas pueden ser síncronas o asíncronas.

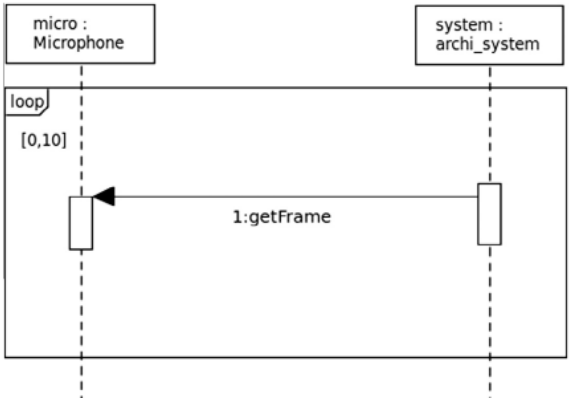


Figura 41 Diagrama de secuencia que modela la interacción sistema-componente de entorno

La Figura 41 muestra un ejemplo de diagrama de secuencia que captura la interacción entre el componente "Microphone" y el sistema del ejemplo de la sección *Exploración del Espacio de Diseño* del capítulo *Ejemplos de Uso y Resultados*. El mensaje "getframe" denota la llamada al servicio para obtener una trama. La figura muestra una sección que es identificada con "loop" lo que denota que se hacen 10 llamadas a ese servicio. En este caso, la llamada es síncrona.

En el escenario también puede intervenir varios componentes. La Figura 42 muestra la interacción del sistema con los componentes "Transmitter" y "Receiver", que refleja una estructura de lazo cerrado, donde la salida del trasmisor está conectada con la entrada del receptor. La Figura 41 muestra un orden entre los mensajes: el componente receptor no mandará una trama hasta que la trama previa que haya sido enviada por el transmisor.

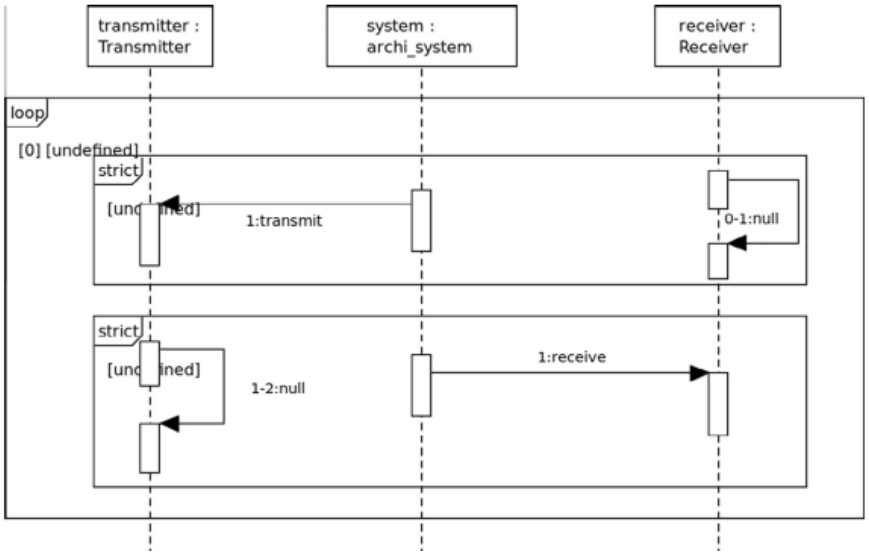


Figura 42 Diagrama de secuencia que modela la interacción sistema-componentes de entorno

Todo esto se puede ver en más en detalle en los trabajos [b)] y [j)].

Este trabajo, así como el ejemplo que se presenta, se realizó con la colaboración de F. Herrera y H. Posadas.

8.2 Acceso a periférico

En un segundo caso, el comportamiento de dicho entorno es representado como archivos de código en los cuales ese comportamiento está implementado. Aquí se pueden dar dos escenarios. En un primer escenario, esos archivos implementan un conjunto de "test-bench" que permiten la validación del sistema [j)], actuando como emuladores del dispositivo del entorno (Figura 43).

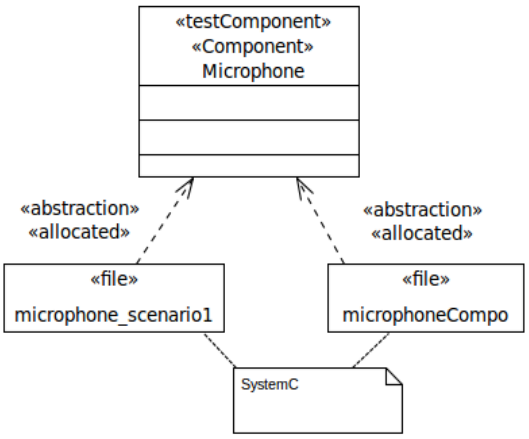


Figura 43 Asociación de archivos a componentes de entorno para simulación

En otro escenario, esos archivos implementan la funcionalidad necesaria para permitir en acceso por parte del sistema a un periférico (por ejemplo, una cámara o un micrófono) como puede verse en [j)] (Figura 44).

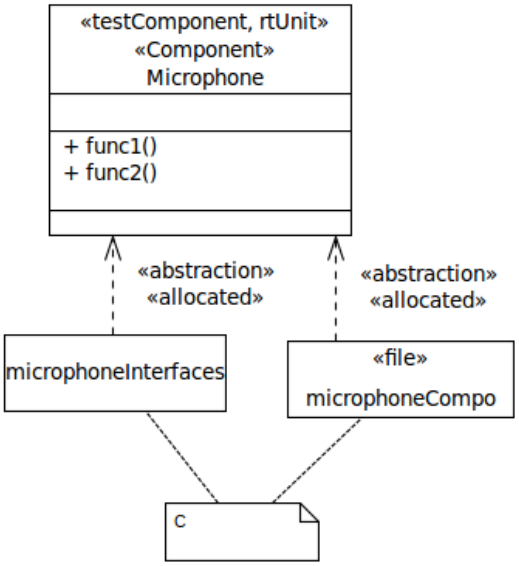


Figura 44 Asociación de archivos a componentes de entorno para implementación

Este trabajo se realizó colaborando con A. Nicolás, H. Posadas y F. Herrera.

8.3 Google Test

La metodología de modelado del entorno se enriqueció con la capacidad de poder modelar Google test [25], colaborando con P. Sánchez y, principalmente, con A. Díaz y Javier González-Bayón. Google test es un entorno basado en C++ que permite la validación de funcionalidad mediante aserciones y que puede ser usado en multitud de plataformas (Linux, Mac OS X, Windows, Cygwin, Windows CE y Symbian). Google test proporciona un gran conjunto de aserciones, permite al usuario definir las suyas propias, definir fallos críticos y no críticos…

La metodología desarrollada permite capturar diferentes tipos de test como "Basic", "Fixture", "Ramdom" y "BVA" ("Boundary-Value Analysis"):

- "Basic": este tipo únicamente incluye un test donde se hacen la inicializaciones, se especifican las funciones a testear y las aserciones.
- "Fixture": este test incluye una clase, donde se especifica el proceso de inicialización del test. Este test es normalmente usado para los casos en los que se desee hacer varios test.
- "BVA": el test "Boundary Value Analysis" permite verificar directamente intervalos concretos relevantes para analizar el rendimiento del sistema.
- "Random": en este test, una entrada del código testeado puede ser llamado de forma aleatoria. El rango de los valores aleatorios y el número de veces que el código es aleatoriamente testeado se especifica.

A partir de estos modelos de test, se genera automáticamente la estructura del código C++ asociados a ellos, donde se incluye no solo el tipo de test, sino las características más importantes de cada uno de ellos (función a testear, rango de valores a considerar en el test, las aserciones, si es fatal o no…). Estos archivos se asocian con los archivos funcionales de la aplicación pudiéndose desarrollar todo el proceso de validación de la funcionalidad.

Los diferentes tipos de test son modelados como paquetes UML. En la Figura 45 se pueden ver como son modelados los diferentes tipos de Google test, donde "GaScenario" es un concepto de MARTE para denotar que en esa carpeta se va a modelar un escenario de análisis.

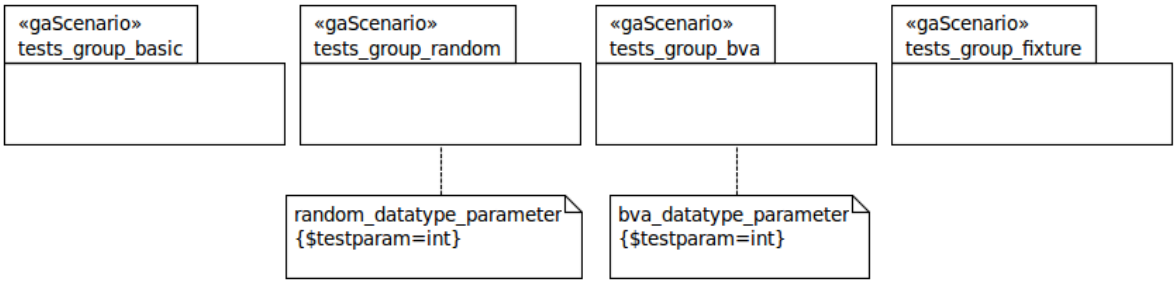


Figura 45 Tipos de Test

Estos escenarios de análisis incluyen la definición de clases, donde se especifica el tipo de test que se va a hacer, y el conjunto de funciones del código funcional que ese test va a verificar (Figura 46).

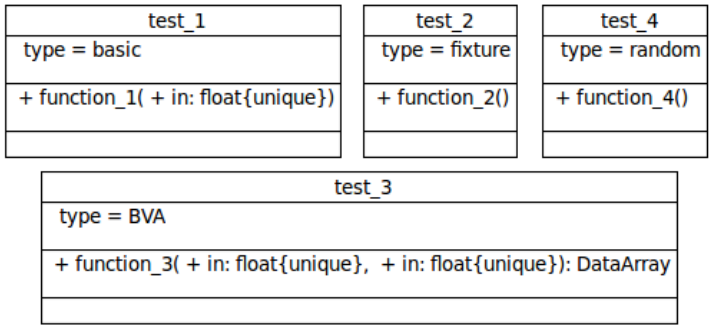


Figura 46 Diferentes tipos de Test de Google

Básico

Este tipo de test tiene asociado dos tipos de características. La primera propiedad es si el test es crítico o no crítico. Si el test es crítico, esto implica que en el caso que el test no tenga éxito, el proceso entero finaliza. En el otro caso, dicho proceso de test no finaliza aunque el test haya reportado un error.

La segunda característica es la expresión lógica. Esta expresión especifica los valores concretos de los parámetros de la función a testear y asocia un valor con un operador lógico:

- Factorial(10)>200;
- Correlation(1)<=-1;

Estos aspectos se capturan mediante variables de modelado. La primera, "$fatal" es una expresión booleana que denota si es crítico o no el test. La segunda variable,

"$expression", denota la expresión lógica. Los operadores lógicos considerados son: >,>=, <, <=, =, !=. La expresión se compone del nombre de la función a testear, los valores de sus parámetros y el operador lógico con el valor asociado (Figura 47):

$expression: maximunCalculation (10.23, 34, 67, 0, 45.67) <=290.45)

Otro caso de expresión lógica considera es la verdadero/falso; cuando la expresión a validar en el test es verdadera o falso:

$expression: InPrimitive (0)=true

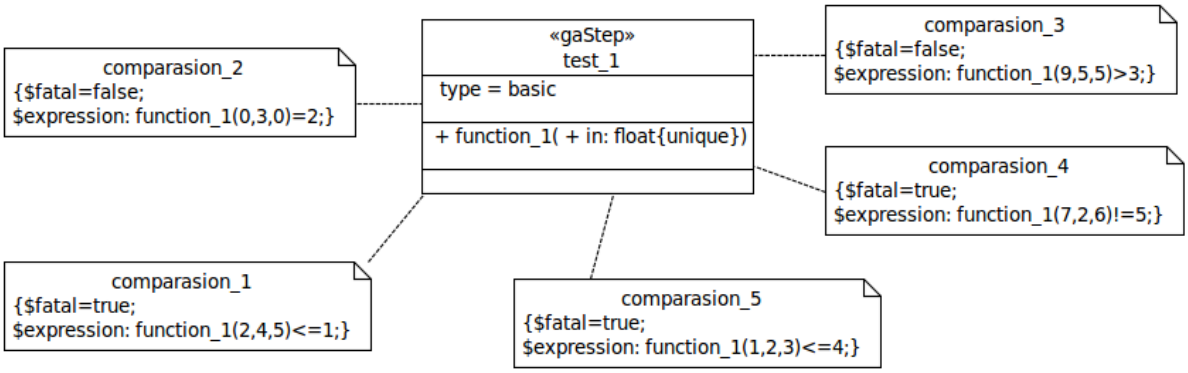


Figura 47 Test Básicos

Aleatorio y BVA

En este tipo de test, se han de anotar el rango de los valores que van a actuar como entradas del test (expresados en la Figura 48 por la variable de modelado "$inputRange").

En el caso de los test "Random" hay que expresar el número de veces que el test es ejecutado (expresados en la Figura 48 por la variable de modelado "$numTests").

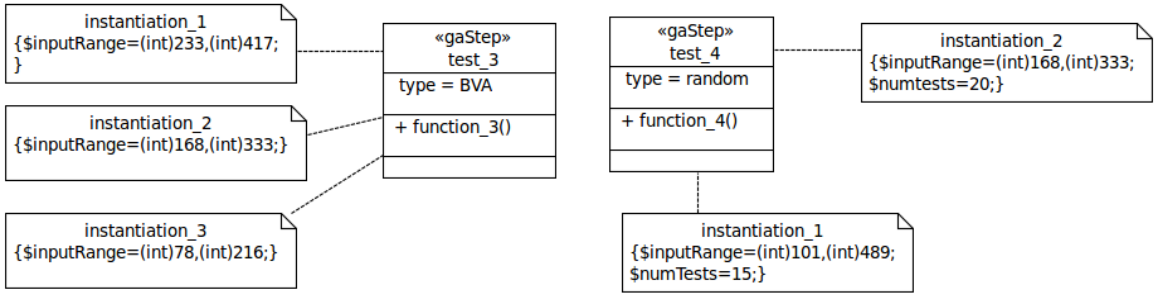


Figura 48 Ejemplos de test "Random" y BVA

Como se puede ver en la Figura 45, en estos tipos de test se ha de especificar el tipo de dato asociado (mediante la variable de modelado "$testparam").

Proyectos en los que estos trabajos se desarrollaron:

1. *COMPLEX, [32], [xx)]. Director principal: E. Villar; presupuesto: 325588€; periodo: 2009-2013.*
2. *PHARAON, [g)] y [aaa)]. Director principal: E. Villar; presupuesto: 354000€; periodo: 2011-2014.*
3. *CRAFTERS, [bbb)]. Director principal: P. Sánchez; presupuesto: 310000€; periodo: 2012-2015.*

Publicaciones resultantes:

1. *Model-Driven Methodology for the Development of Multi-level Executable Environments, [j)].*
2. *A Model-Driven Methodology for the Development of SystemC Executable Environments, [hh)].*

8.4 Ataques a Redes

Una parte del modelado de la redes es el desarrollo de todo los mecanismos expresivos para poder capturar y modelar diferentes tipos de atacantes, con el fin de poder crear escenarios que validen la seguridad de la red contra interferencias externas a ellas. Este trabajo se realizó colaborando con P. Sánchez y, principalmente, con A. Díaz y J. Medina.

Para poder realizar el modelado de los atacantes desarrollé un nuevo conjunto de elementos expresivos que, basados en el elementos de MARTE ya definidos, pudieran capturar los diferentes tipos de ataques considerados en este trabajo. La Figura 49 muestra la definición de estos nuevos mecanismos expresivos. Estos serán usados para crear los entornos de evaluación (ejemplo de la sección *Redes de Sensores* del capítulo *Ejemplos de Uso y Resultados*).

Tres grandes grupos de atacantes han sido definidos en esta metodología en función de lo que se presentó en el trabajo [87]. En las siguientes secciones se van mostrar unos ejemplos de los tipos de atacantes que se pueden modelar con esta metodología y que serán usados en el caso de uso presentando en la sección *Redes de Sensores* del

capítulo *Ejemplos de Uso y Resultados*. El conjunto completo se pueden ver en el trabajo [a)], donde se explican en más detalle cada una de ellos.

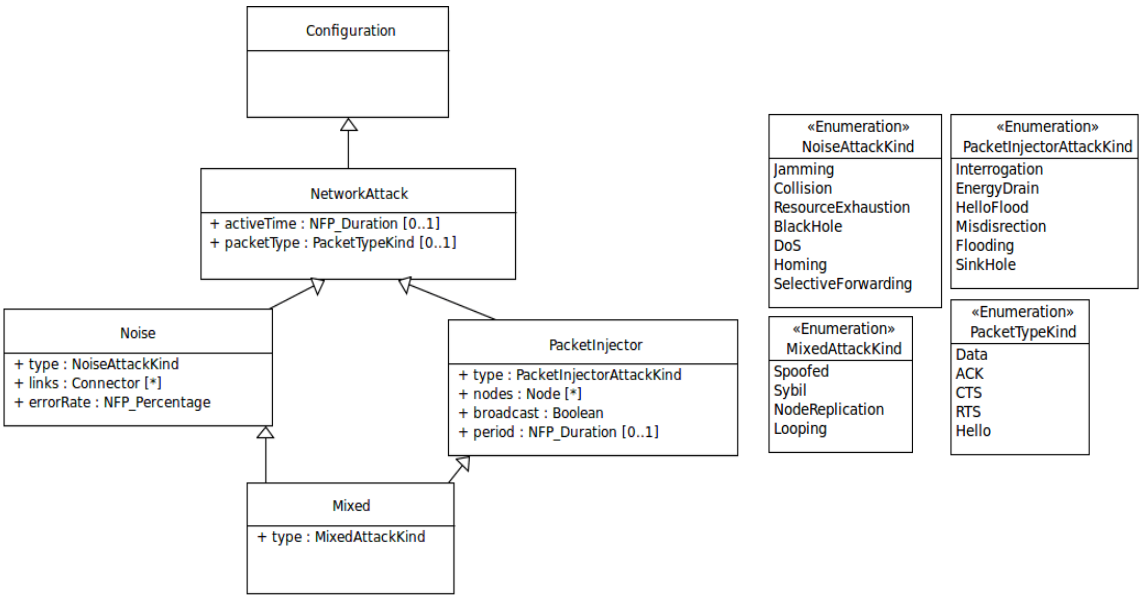


Figura 49 Modelado de los atacantes

Atacantes que introducen ruido en la red

El primer grupo de atacantes se introducen ruido en la red, con el objetivo de incrementar la posibilidad de pérdida de paquetes, modificando la probabilidad de éxito en el envío de paquetes en una conexión nodo-nodo. La lista completa de los atacantes considerados es ampliamente descrita en [a)] por A. Díaz; los nombres pueden ver en la Figura 49 (realizada con la colaboración de J. Medina), en el elemento enumeración llamado “NoiseAttackKind”. Aquí se explicaran aquellos que se utilizarán en el caso de uso incluido en sección *Redes de Sensores* del capítulo *Ejemplos de Uso y Resultados*.

Los atacantes son:

- “Jamming”: este ataque niega el servicio a usuarios autorizados mientras que el tráfico legítimo es atascado por la cantidad abrumadora de tráfico ilegítimo. Este tipo de atacante interrumpe la funcionalidad de la red mediante la emisión de señales de alta energía. Los paquetes afectados por estas señales de alta energía no pueden llegar a su destino, ya que el nodo de destino sólo recibe ruido y no es capaz de decodificar el paquete. Como ejemplo, la Figura 50 (realizada por A. Díaz) ilustra cómo se comporta este tipo de ataque. El atacante produce grandes cantidades de

interferencia intermitente o persistente. Esta interferencia afecta directamente a, al menos, 3 nodos de red (B, D y F).

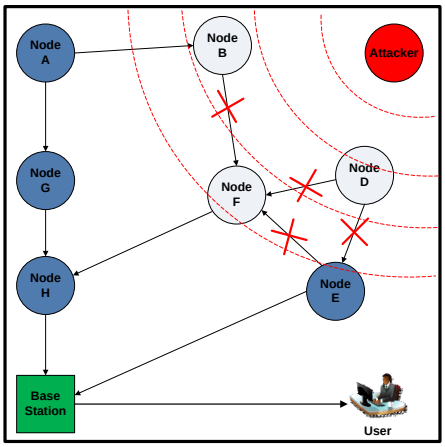


Figura 50 Esquema del atacante "Jamming"

- Ataque por colisión: aparece cuando un nodo atacante no sigue el protocolo de control de acceso al medio, produciendo colisiones con las transmisiones del nodo vecino, enviando un paquete corto y ruidoso. Los paquetes chocan cuando dos nodos intentan transmitir a la misma frecuencia, simultáneamente, produciendo corrupción de paquetes.

Atacantes que introducen paquetes en la red

Otro tipo de ataques son los llamados ataques de inyección de paquetes falsos. Estos introducen paquetes en la red con el objetivo de hacer que los nodos originales los procesen, aumentando el tráfico en la red y, por lo tanto, congestionándolo. La lista completa de los atacantes considerados en esta metodología es ampliamente descrita en [a)] por A. Díaz; los nombres pueden ver en la Figura 49, en el elemento enumeración llamado "PacketInjectorAttackKind". Aquí se explica el que se utilizará en el caso de uso incluido en sección *Redes de Sensores* del capítulo *Ejemplos de Uso y Resultados*.

El atacante es:

- "Hello Flood": el atacante, normalmente, intenta drenar la energía de un nodo o agotar sus recursos. Un atacante con gran poder de transmisión transmite paquetes "HELLO" (usados en muchos protocolos para detección de nodos en la red) para convencer a cada nodo de la red de que hay un nuevo nodo en la red (que sería

el atacante), dañando un gran número de nodos al perder energía enviando paquetes a este nuevo vecino.

Atacantes que introducen ruido y paquetes en la red

Finalmente, algunos ataques (ataques mixtos) causan problemas en la red, mezclando los efectos de los anteriores tipos de atacantes. Estos ataques hacen que la red pierda algunos paquetes, pero al mismo tiempo introducen nuevos paquetes en la red. Por lo tanto, estos ataques alteran los tipos de paquetes transmitidos, reduciendo el número de algunos tipos de paquetes pero aumentando otros, con el consecuente impacto en el rendimiento de la red.

La lista completa de los atacantes considerados es ampliamente descrita en [a)] por A. Díaz; los nombres pueden ver en la Figura 49, en el elemento enumeración llamado "MixedAttackKind". Aquí se explicara el que se utilizará en el caso de uso incluido en sección *Redes de Sensores* del capítulo *Ejemplos de Uso y Resultados*.

El atacante es:

- "Looping in the Network": este ataque consiste en la modificación del enrutamiento de la red al afectar a la transmisión de datos del nodo. Un nodo "falso" informa a otro nodo que es el nodo final de la cadena o que es el nodo más cercano al final de la cadena de nodos. Este nodo "falso" vuelve a enviar el paquete a la red, encerrando el paquete en un bucle infinito (Figura 51, realizada por A. Díaz).

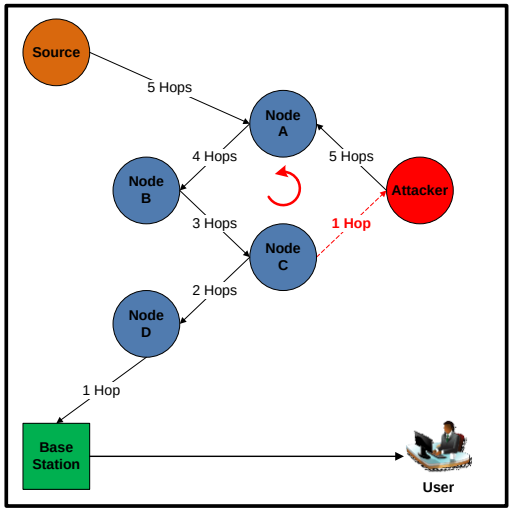


Figura 51 Esquema del atacante "Looping in the Network"

Proyectos en los que este trabajo se desarrolló:

1. *TOISE, [yy)]. Director principal: P. Sánchez; presupuesto: 163861€; periodo: 2011-2013.*

Publicaciones resultantes:

1. *High-Level design of wireless sensor networks form performance optimization under security hazards, [a)].*
2. *Modeling and Simulation of secure wireless sensor network, [r)].*

9. Sistemas Críticos y de Criticidad Mixta

Los sistemas críticos son aquellos en los que un fallo puede tener repercusiones graves. Para abordar el diseño de estos sistemas es necesario definir y verificar una serie de propiedades no funcionales que van asociadas a cada uno de los componentes o al sistema en su conjunto.

La criticidad mixta es una propiedad que aparece en sistemas de alta complejidad, una vez que integran más y más funcionalidades. Las funcionalidades integradas son de naturaleza diferente y con diferentes requerimientos de rendimiento, eficiencia energética y coste. Por otra parte, los requisitos de seguridad comienzan a jugar un papel más relevante, necesitando nuevas restricciones asociadas a las diferentes propiedades que no tienen la misma criticidad, en términos de impacto sobre el sistema.

En la metodología de modelado desarrollada se ha trabajado principalmente en el manejo de criticidades múltiples, como caso general de sistemas críticos y de criticidad mixta. Como resultado, la metodología propuesta permite capturar los siguientes aspectos de la criticidad mixta:

- Asociación a una propiedad extrafuncional de un conjunto de valores potencialmente diferentes, cada uno correspondiente a un determinado nivel de criticidad.
- Asociación de un nivel de criticidad a requisitos y contratos.
- Asociación de un nivel de criticidad directamente con un elemento del modelo, por ejemplo un componente de aplicación, o un recurso de plataforma.

El primer caso hace referencia a la extensión a análisis de tiempo real a sistemas de criticidad mixta. Por ejemplo, se han desarrollado nuevos análisis de planificación que requieren la asociación de varias figuras del WCET (“Worst-Case Execution Time”) a una misma tarea. Cada valor de WCET corresponde a un nivel de criticidad dado.

El segundo caso se fundamenta en el hecho de que no todos los requisitos y/o contratos tienen la misma importancia en el diseño del sistema. La importancia de cada restricción/contrato depende del sistema y del escenario de diseño. Obsérvese que la asociación de los niveles de criticidad a los requisitos es bastante genérica. Las criticidades pueden referirse a requisitos del mismo tipo de restricciones (por ejemplo, restricciones de tiempo, como plazos y el “throughput”) o no (por ejemplo, una latencia y el consumo de energía); y para diferentes aplicaciones o no.

La necesidad de asociar criticidades directamente a los elementos de modelo obedece a razones prácticas a la hora de hacer el modelo. Cuando se asocia una criticidad a un componente de aplicación, todas las propiedades de elemento heredan la criticidad del elemento. La asociación de una restricción de criticidad a un componente de plataforma, es decir, un recurso de la plataforma, es también práctica, una vez que los estándares de seguridad utilizan un nivel de criticidad para imponer requisitos sobre cómo se debe desarrollar ese componente.

Por tanto, el modelo de criticidad mixta consiste:

- Anotaciones de valores a propiedades no funcionales.
- Restricciones sobre propiedades no funcionales.
- Caracterizar a componentes de la aplicación y de la plataforma.

Este trabajo se fundamentó en la extensión que realizó J. Medina al perfil de MARTE para poder capturar niveles de criticidad a elementos y propiedades del modelo (Figura 52, realizada por J. Medina).

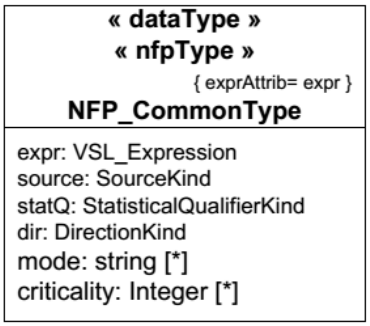


Figura 52 Extensión de MARTE para capturar niveles de criticidad

El resto de trabajo colaboré en la labor realizada por F. Herrera y J. Medina.

9.1 Propiedades extra-funcionales

Para ilustrar la necesidad de especificar niveles de criticidad a propiedades no-funcionales, se va a hacer uso del análisis de planificabilidad. En técnicas clásicas de análisis de planificabilidad, el tiempo de ejecución del peor caso (WCET) es una propiedad no funcional típica asociada a una función o tarea. Algunas metodologías requieren la asociación de una lista de WCETs a cada tarea. En este caso, lo que distingue cada WCET de la lista es que le corresponde un nivel de criticidad específico. Para estos

casos, se requiere de la capacidad de asociar niveles de criticidad a anotaciones de valores de propiedades no funcionales.

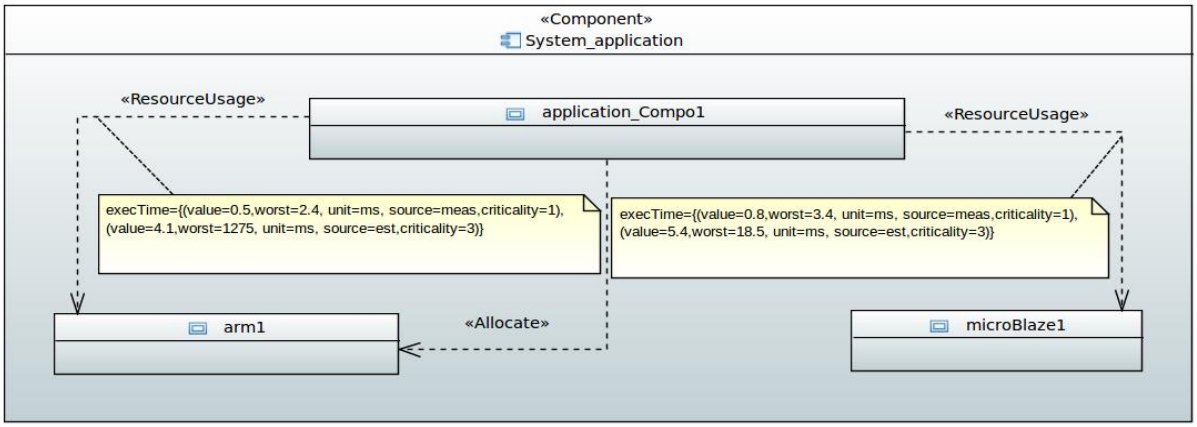


Figura 53 Criticidad asociada de valores de tiempos de ejecución

Además, hay escenarios donde puede ser interesante la asociación del mismo requisito a varios elementos de modelado. Un ejemplo de este caso se muestra en la Figura 55. En dicha figura se muestra un caso en el que la misma restricción con la misma criticidad afecta a varias instancias de modelado, sobre una propiedad extrafuncional concreta, que es, en este caso, el WCET.

Para más información ver [tt)].

9.2 Criticidades para restricciones en propiedades no-funcionales

Una de las principales necesidades de modelado es poder soportar a la especificación de los niveles de criticidad para los requisitos y contratos sobre propiedades extrafuncionales.

Los requisitos pueden entenderse como expresiones booleanas que pueden referirse tanto a las propiedades funcionales como a las extrafuncionales. Esto último, es decir, el soporte de requisitos sobre las propiedades extrafuncionales (denominadas no funcionales en la especificación MARTE), proporciona mecanismos de análisis para propiedades, por ejemplo, relacionadas con el tiempo, el consumo de energía, la temperatura y la fiabilidad. Esto posibilita la asignación de diferentes "niveles de importancia" a los diferentes requisitos extrafuncionales.

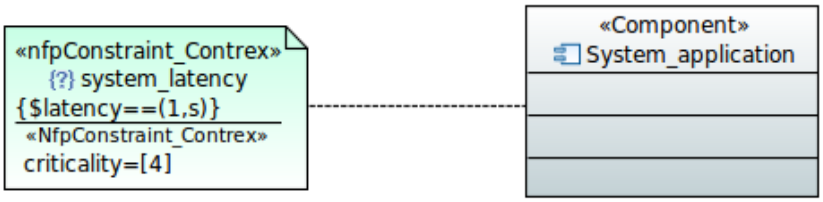


Figura 54 Criticidad asociada a una restricción de rendimiento

La Figura 54 muestra un ejemplo en el que la criticidad se aplica a una expresión explícita de una restricción del modelo. La expresión ("latencia == (1, s)") refleja un requisito sobre la latencia del sistema. El nivel de criticidad muestra que nivel de importancia que tienen que esa condición del diseño se debe cumplir

Otro ejemplo se muestra en la Figura 55; hay escenarios donde puede ser interesante la asociación del mismo requisito a varios elementos de modelado. En dicha figura se muestra un caso en el que la misma restricción con la misma criticidad afecta a varias instancias de modelado, sobre un propiedad extrafuncional concreta, que es, en este caso, el WCET.

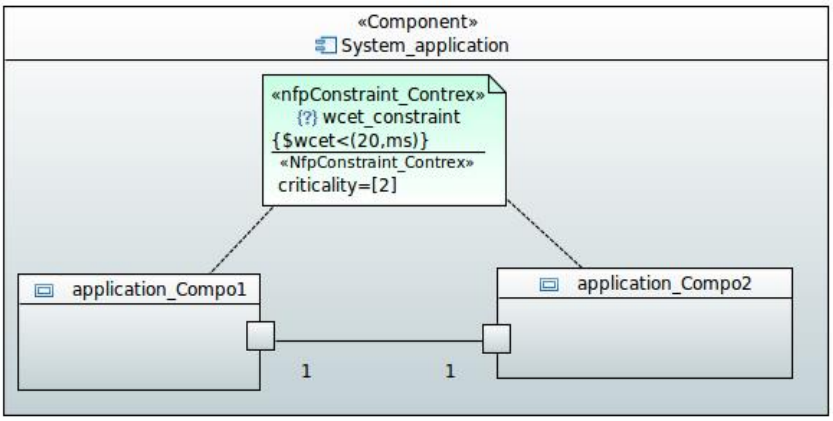


Figura 55 Criticidad asociada a varios elementos

9.3 Criticidades para componentes del sistema

Otra necesidad de modelado es la asociación de los niveles de criticidad directamente a los elementos del modelo, como componentes de la aplicación o de la plataforma.

Se han identificado al menos dos casos de interés. Un primer escenario es el mapeo de los elementos de la aplicación a los recursos de la plataforma. Es decir, en ciertos contextos, se espera que las metodologías de diseño permitan asignar niveles de criticidad tanto a los componentes de la aplicación como a los recursos de la plataforma. En el primer caso, la criticidad de los componentes de la aplicación representaría una criticidad "requerida". En este último caso, la criticidad tendría una semántica más relacionada con la criticidad que el recurso de plataforma puede soportar, estos es, solo es posible asignar componentes de aplicación a ese recurso que tengan menor o igual nivel de criticidad. A partir de esta información, se necesitan los análisis que comprueben la coherencia del mapeo, comprobando si un componente de aplicación determinado se puede asignar a un componente de la plataforma, atendiendo a las criticidades de ambos.

Un requisito a la hora de desarrollar una metodología de modelado poder crear los modelos con criticidad de una manera lo más simple posible. Este hecho permite la construcción de modelos más sintéticos y metodologías de modelado más ágiles, siempre que los niveles de criticidad asignados correspondan a restricciones\contratos bien conocidos, que puedan derivarse implícitamente de la asociación con el elemento de modelado.

En el caso de la Figura 56, el nivel de criticidad está asociado a un componente de aplicación, de esta forma, el propio elemento y todos sus atributos asociados tendrían ese nivel de criticidad.

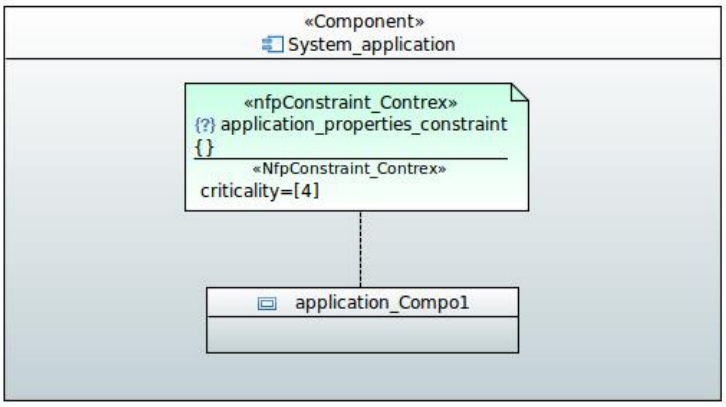


Figura 56 Criticidad asociada a un componente de aplicación

Proyectos en los que este trabajo se desarrolló:

1. *CONTREX, [ww)]. Director principal: E. Villar; presupuesto: 391336; periodo: 2013-2016.*

Publicaciones resultantes:

1. *UML-Based Single-Source Approach for Evaluation and optimization of Mixed-Critical Embedded Systems. [s)].*
2. *UML/MARTE Modelling for Design Space Exploration of Mixed-Criticality Systems on top of Time-Predictable HW/SW Platforms. [u)].*
3. *A model-based, single-source approach to design-space exploration and synthesis of mixed-criticality systems*.[w)]

Capítulo V

Entorno de Desarrollo: Integración de Herramientas

Como fue introducido en la sección *Metodología de Modelo Único*, el entorno de desarrollo creado durante esta tesis incluye un gran número de herramientas y lenguajes, lo cuales fueron mostrados en la Figura 8.

En sucesivos apartados se van a ir explicando en detalle la mayor parte de ellas.

Lo primero a presentar es la interrelación (que fue introducido en la Figura 9) entre las diferentes herramientas y lenguajes que forman parte del entorno de desarrollo, que se presenta en esta tesis, lo cual es mostrado en la Figura 57.

De forma introductoria, desde el modelo de UML/MARTE, se generan una serie de archivos mediante los generadores de código implementados. El primer grupo de estos se genera en SystemC y sirve para realizar simulaciones funcionales. Una de estas generaciones está formalmente soportada por ForSyDe, como se explicó en la sección *Formalización de UML/MARTE: ForSyDe* del capítulo *Modelado de Sistemas Electrónicos*.

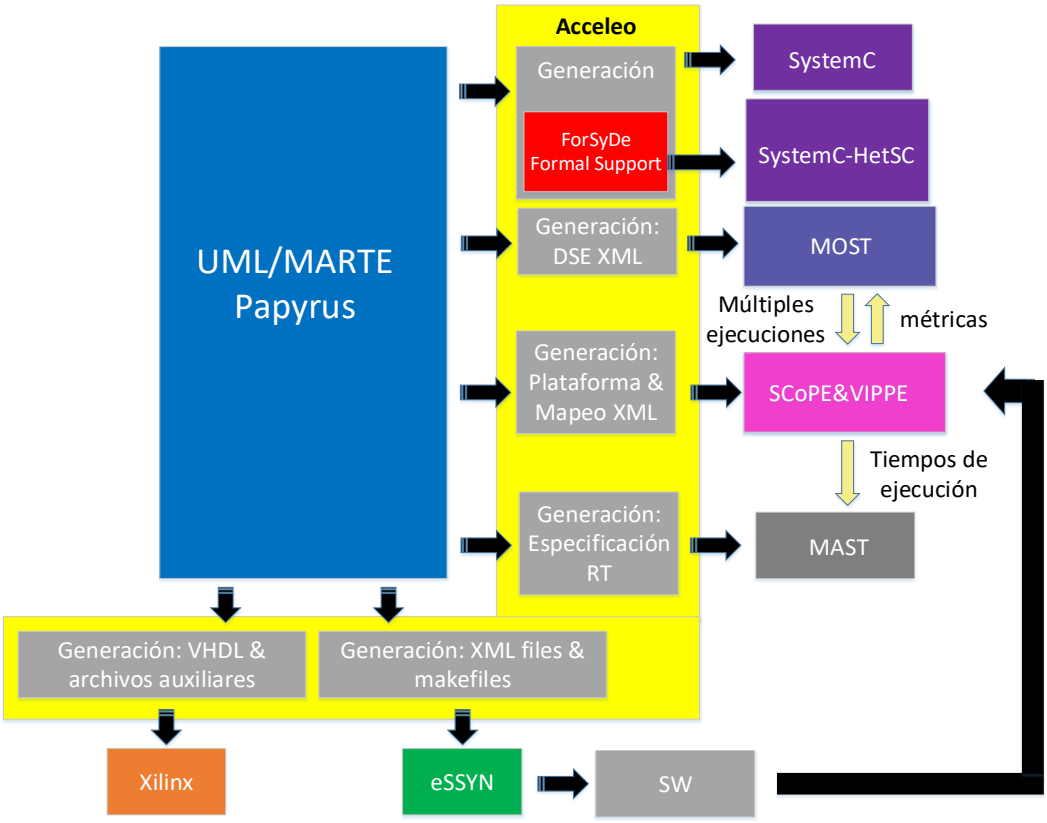


Figura 57 Relación de herramientas

Otro tipo de generación produce archivos XML que sirven para trasmitir la información capturada en el modelo a un conjunto de herramientas. Finalmente, en el caso del análisis de planificabilidad se genera un archivo de texto con toda la información de tiempo real necesaria para dicho análisis.

1. Infraestructura base

Uno de los objetivos de esta tesis es la de demostrar las capacidades de la metodología propuesta proporcionando un entorno de desarrollo totalmente integrado, donde el conjunto de herramientas que se van a poder utilizar estén totalmente integradas. Como muestra la Figura 8 y Figura 57, son variadas en número y naturaleza las herramientas a integrar. Por ello se requiere que este entorno permita una integración lo más fácil posible, así como la posibilidad de realizar interfaces de usuario lo más intuitiva posible. Con esta premisa, como entorno de integración se usó Eclipse [16].

Eclipse integra las herramientas para crear el modelo así como aquellas que son necesarias para leer la información captura en dicho modelado y proporcionársela al conjunto de herramientas para poder ser ejecutadas. De la misma manera, todas las herramientas desarrolladas durante esta tesis, tales como como generadores de código, nuevos elementos expresivos (estereotipos y librerías de elementos de modelado), lanzadores de herramientas externas, compilación… se incluyen en "plugins" que se instalan en Eclipse.

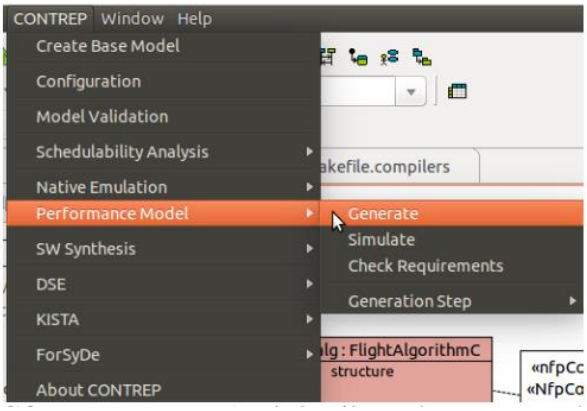


Figura 58 Ejemplo de menú de "plugin"

Finalmente, para poder ejecutar toda esta batería de herramientas, se desarrolló un interfaz de usuario mediante menús, los cuales permiten ir lanzando cada una de las herramientas del entorno de diseño (sirva como ejemplo la Figura 58) de una manera fácil.

Una de las tareas realizadas en esta tesis fue la de la implementación de toda la infraestructura necesaria para lanzar cada una de las herramientas que componen el entorno. Además, se integraron todos los correspondientes generadores de código y todos aquellos métodos que permitieran compilar y ejecutar el código funcional generado. Implementado en Java, este entorno fue completamente incluido en Eclipse mediante un conjunto de "plugins", el cual también fue implementado por mí.

2. Modelado y Transformación del Modelo

La herramienta de modelado que se ha utilizado es Papyrus (sección *Otros Lenguajes y Herramientas Usados en la Tesis* del capítulo *Estado del Arte*). Papyrus permite crear modelos UML y aplicar un conjunto de perfiles estándar, entre ellos MARTE, con el fin de enriquecer los modelos UML con las propiedades expresivas que poseen dichos perfiles.

Además de esto, Papyrus también permite la creación de nuevos elementos expresivos (como estereotipos y elementos de librerías) lo cual ha permitido cubrir las carencias expresivas que se ha ido detectando en MARTE a la hora de abordar las diferentes áreas de aplicación. Como ejemplos de esto, los estereotipos para denotar las vistas ([oo)]), para capturar y especificar los canales ([oo)])… conforman un perfil que permite cubrir aquellas necesidades expresivas que tanto UML como MARTE no poseen. Además, Papyrus también permite crear librerías de elementos de modelado, con lo que el diseñador puede crear librerías de componentes (por ejemplo un tipo de nodo de red con su estructura interna, o un componen HW definido por un conjunto concreto de propiedades…) y ser cargadas en un nuevo modelo, sin tener que volver a especificar el mismo elemento en modelos diferentes.

Papyrus es una herramienta de modelado y carece de otras capacidades necesarias para desarrollar los trabajos aquí presentados. Más concretamente, a la hora de generar código a partir de dichos modelos, Papyrus no lo permite directamente. La información de la modelo capturada con Papyrus es guardada en un conjunto de archivos auxiliares de la propia herramienta. Estos archivos pueden ser leídos por otras herramientas. Para este fin, se utiliza la herramienta Acceleo (sección *Otros Lenguajes y Herramientas Usados en la Tesis* del capítulo *Estado del Arte*) que permite la transformación de modelos UML/MARTE a texto mediante la utilización del lenguaje "Model To Text Transformation Language" (MOFM2T) [15]. Este entorno permite la transformación del modelo en el correspondiente conjunto de archivos de texto, que algunas veces serán archivos de código SystemC, otros C, otras veces XML…, dependiendo de la tarea que se esté realizando y/o de la herramienta auxiliar con la que se quiera enlazar.

En cuanto a la implementación del entorno de desarrollo (a parte de la metodología UML), una gran parte de la labor realizada en el contexto de esta tesis fue la del desarrollo e implementación del conjunto de generadores que transformaran la información del modelo en según qué archivos de texto. Todo esto llevó a la necesidad de aprender el MOFM2T y a saber cómo Papyrus guardaba la información en esos archivos auxiliares.

3. Análisis: Simulación funcional

En esta tesis, la simulación es entendida como un proceso en el que el modelo UML/MARTE, que no puede ser ejecutado directamente, es transformado a un lenguaje de acción que sí es ejecutable. Este hecho requiere de un mapeo entre los elementos del modelo UML/MARTE y las estructuras y elementos de dicho lenguaje de acción con el fin de habilitar dicha transformación. De esta forma, el diseño podrá ir verificando la validez de su diseño.

Durante el proceso de transformación, la información capturada en el modelo UML/MARTE debe ser traslada a la especificación ejecutable sin ningún perdida de información. Por esta razón, es un requisito indispensable establecer un mapeo unívoco entre los elementos de UML/MARTE utilizados para la realización del modelo, con el lenguaje de acción.

Aquí se van a distinguir dos situaciones; la primera basada en lo mostrado en la subsección *Formalización de UML/MARTE: ForSyDe* del capítulo *Modelado de Sistemas*, la simulación que se llevará a cabo es puramente funcional, donde ninguna noción de temporalidad es tenida en cuenta: la única relación existe es la causalidad entre eventos ligados entrada-salida. De esta forma se puede estudiar nuestro sistema, que en principio es entendido como un sistema concurrente, y ver si presenta algún tipo de indeterminismo.

El otro escenario considerado es una simulación funcional donde sí existe una noción de tiempo físico, lo que hace establecer un ordenamiento más global de los eventos que se van dando en dicha simulación.

En ambas situaciones, el lenguaje de acción es SystemC.

3.1 Modelo Atemporal

En la parte de simulación funcional, hay varias estrategias que se pueden seguir. La primera de ellas es en la que la aplicación es concebida como un sistema atemporal donde ninguna noción sobre el tiempo es considerada, solamente eventos que representan datos que se transmiten.

Relación UML/MARTE-SystemC

Tomando como base la relación UML/MARTE-ForSyDe descrita en la sección *Formalización de UML/MARTE: ForSyDe* del capítulo *Modelado de Sistemas Electrónicos*, se estableció una mapeo entre UML/MARTE y SystemC para la generación automática de especificaciones ejecutables.

En [i)], [j)], [n)] y [jj)] se describe la interrelación entre UML/MARTE-SystemC usando ForSyDe con un enlace formal que asegura la preservación semántica en el proceso de transformación. Esto tiene relación con la estructura comunicación-cómputo del sistema, así como de las propiedades asociadas a dicha estructura, que definen el comportamiento del sistema. Las reglas de transformación son soportadas formalmente con ForSyDe, que actúa como modelo de referencia. Un mismo modelo ForSyDe puede ser inferido desde el modelo UML/MARTE y desde la especificación SystemC, manteniendo la coherencia semántica entre ambos. Preservar dicha coherencia semántica hace que la transformación UML/MARTE-SystemC sea correcta por construcción.

Hay aspectos en la relación UML/MARTE-SystemC que no pueden ser formalmente asociados y que deben ser definidos metodológicamente, como por ejemplo elementos relacionado con la estructura como puertos, módulos... En [e)] se describe la metodología que, basada en modelo UML/MARTE, permite generar especificaciones ejecutables en SystemC, donde módulos, puertos y procesos son obtenidos a partir de elementos del modelo.

Además de los aspectos estructurales, la metodología permite capturar canales de comunicación con propiedades específicas de un conjunto de MoCs [e)]. Para poder implementar estos MoC se usó la librería HetSC (sección *Otros Lenguajes y Herramientas Usados en la Tesis* del capítulo *Estado del Arte*) que proporciona canales SystemC que capturan las propiedades concretas de MoCs (se explicará en la siguiente sección).

Los modelos de alto nivel son esenciales para el diseño de ESL ya que permiten un modelado rápido y una rápida simulación de los mismos, haciendo que el proceso de exploración del espacio de diseño sea factible, permitiendo tomar decisiones óptimas en las primeras fases del diseño. El lenguaje SystemC es el dominante para ESL, en lo que a especificación, modelado y diseño se refiere.

El hecho de poder combinar SystemC-ESL con MARTE-MDA hace que se aprovechen las ventajas de ambos mundo. Las diferentes vistas de un modelo de MARTE

pueden reflejar diferentes refinamientos y aspectos del sistema, los cuales pueden ser validados, produciendo una especificación ejecutable SystemC 'equivalente' en términos funcionales.

La Figura 59 muestra la primera aproximación de cómo se va a realizar este mapeado. En el modelo MARTE, se van a capturar la estructura de nuestro sistema, en forma de elementos concurrentes conectados mediante canales. Además de esto, mediante diagramas de actividad, se capturarán la funcionalidad de dichos elementos concurrentes. Desde aquí se establece un mapeado a elementos SystemC, para obtener el código ejecutable del modelo de MARTE. Esta transformación se sustenta mediante ForSyDe, lo que proporciona la consistencia formal a dicha transformación.

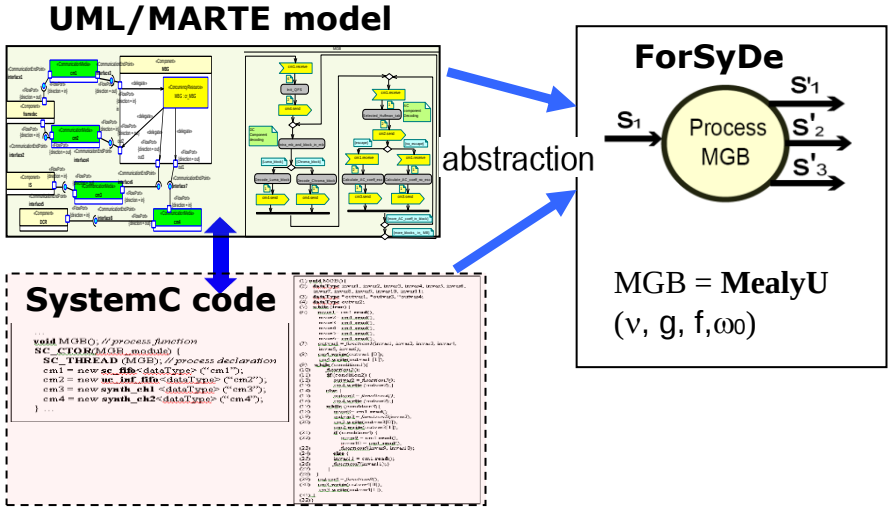


Figura 59 Enlace Formal MARTE-SystemC

El enlace entre MARTE-SystemC consiste en una abstracción ForSyDe común a ambos. Esto requiere la integración coherente de las consideraciones de [33] y de [e)] para la formalización de MARTE y de SystemC, respectivamente, para establecer las posibilidades y límites de la relación MARTE/SystemC.

Una primera consideración es que el enlace formal establecido por ForSyDe hace referencia a la ejecución semántica. Por tanto, el primer paso de la abstracción consiste en eliminar de los modelos MARTE y SystemC aquella información que no tiene impacto en dicha ejecución semántica. Esta eliminación consiste en aquellos elementos cuya misión consiste de dotar al sistema de jerarquía estructural: puertos y componentes estructurales en MARTE; y módulo, puertos y diferentes tipos de enlaces en SystemC. Esto implica que la correspondencia uno-a-uno en la relación entre los elementos usados

para especificar estructura jerárquica es irrelevante cuando se establece el mapeo MARTE-SystemC. Por tanto, para proporcionar la coherencia semántica en la relación MARTE/SystemC, el modelo ForSyDe se debe enfocar en la estructura concurrente y de comunicación. El proceso ForSyDe refleja el elemento concurrente de MARTE y el proceso SystemC.

La comunicación está dividida en dos casos, en función al tipo de comunicación (como se vio en la sección *Formalización de UML/MARTE: ForSyDe* del capítulo *Modelado de Sistemas Electrónicos*). Cuando el canal de comunicación es semántica simple, no hay alteración en los valores transmitidos. En este caso, la asociación "communication media" y "channel" SystemC es directa. En MARTE, esto se deduce de las propiedades asociadas al canal, mediante el valor de los diferentes atributes que lo caracterizan. En SystemC, esta semántica se obtiene de la documentación o del propio código de implementación del canal.

Respecto al segundo escenario, canales con semántica compleja (sección *Formalización de UML/MARTE: ForSyDe* del capítulo *Modelado de Sistemas Electrónicos*), ForSyDe proporciona la correspondiente abstracción formal al "communication media" de MARTE y a la correspondiente esquema de comunicaciones de SystemC.

En el caso que como mecanismo de comunicación se use una variable compartida, el correspondiente modelo formal existe, siempre que se preserven un conjunto de condiciones. Estas condiciones pueden establecerse mediante aserciones [33].

ForSyDe también soporta la formalización del comportamiento del proceso, capturado mediante un diagrama de actividad en el modelo MARTE (sección *Formalización de UML/MARTE: ForSyDe* del capítulo *Modelado de Sistemas Electrónicos*), y como código secuencial en el proceso SystemC.

La Figura 60 muestra la abstracción ForSyDe del diagrama de actividad (ya explicado en la sección *Formalización de UML/MARTE: ForSyDe* del capítulo *Modelado de Sistemas Electrónicos*) y su correspondencia con código SystemC. La entradas de datos son accesos a canal, mediante la función auxiliar "read ()","vp=ch_1.read ()". Estos son tantos como datos se desean leer. Luego estos datos leídos se computan ("function_i (vp,…, vq,…, vr,…vs, va,…, vb,…, vc,…vd);"), obteniendo los resultados, que se envían por los correspondientes canales, según la forma "chp_1.write (va);…;".

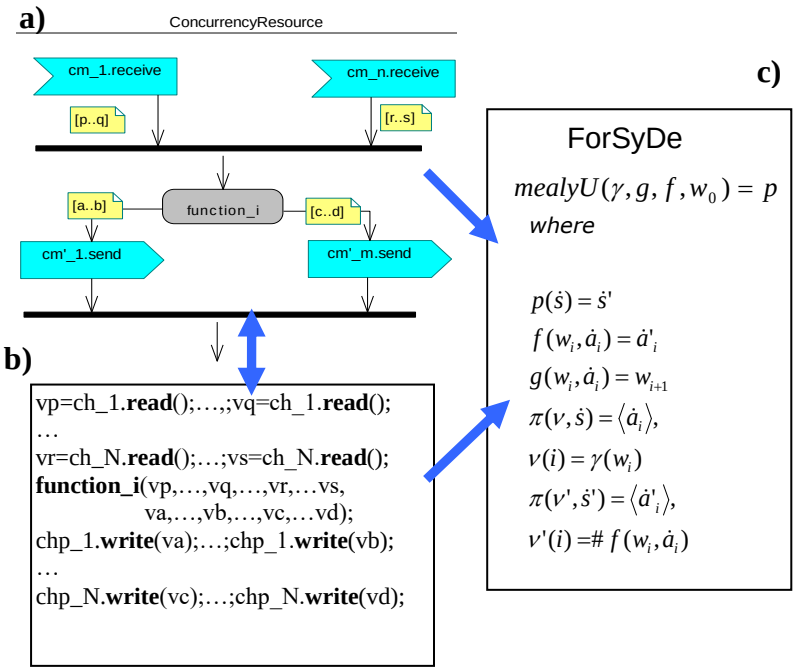


Figura 60 Funcionalidad en MARTE-SystemC

En el caso más general, la abstracción consiste en una máquina de estado Mealy (denotado en la Figura 60 en ForSyDe a través del constructor "mealyU"). El sufijo U denota una semántica ejecutiva atemporal, que es el caso más abstracto en lo que concepto temporal se refiere. El constructor "mealyU" toma como función *f*, *g* como función de siguiente estado, la función γ para definir la partición de las entradas y el estado inicial w_0 como argumento.

Una aplicación adicional de extraer el modelo ForSyDe es la generación de propiedades que la especificación SystemC debe satisfacer bajo cualquier condición dinámica en cualquier test. Se ha de destacar que el modelo ForSyDe es estático en su naturaleza; no incluye la sincronización y mecanismos de disparo usados por el modelo SystemC.

Un caso típico de esto es la comunicación mediante variables compartidas, el cual es un problema bien conocido en ingeniería. Como la semántica de simulación SystemC es no-expulsora, proteger el acceso a variables compartidas no implica ninguna diferencia. En cambio, es un problema de implementación cuando se mapea un proceso a SW o HW.

Según lo definido por E. Villar, una variable compartida implementa correctamente una señal ForSyDe cuando se aplican las siguientes condiciones:

- Cualquier dato escrito por el productor es leído por el consumidor.
- Cualquier dato escrito por el productor es leído una única vez por el consumidor.

En algunos casos, para simplificar el diseño, se puede decidir usar una variable compartida como memoria local. Este problema se puede resolver renombrando la variable. En este caso, una condición se ha de aplicar:

- Si el consumidor usa una variable compartida como memoria local, ningún nuevo dato puede ser escrito por el productor después del último acceso a la memoria local hecha por el consumidor; durante la vida de la memoria local de la variable compartida.

Canales HetSC

La metodología HetSC realizada por F. Herrera proporciona un conjunto de canales que realizar especificaciones SystemC bajo la semántica de un MoC concreto [88].

El modelo de canal para MoC se muestra en la Figura 61, el cual es un componente especificado por un conjunto de elementos MARTE que le dotan de la capacidad necesaria expresiva para capturar las propiedades de varios MoCs. Como forma de facilitar al diseñador su uso, se implementó una librería de componentes de canales, que podrán ser importados a la hora de hacer el modelo (*Modelado y Transformación del Modelo* de este capítulo).

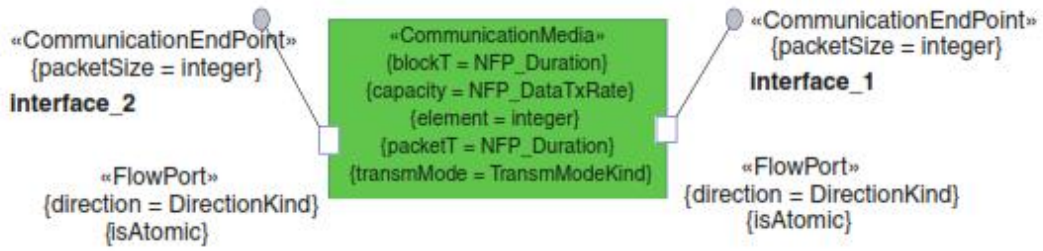


Figura 61 Modelo de canal para MoC

Para ver como se combinan los diferentes valores de los atributos del canal ver las publicaciones [e)] y [mm)]. Estos trabajos se realizaron con la colaboración de F. Herrera

y J. Medina. Con estos atributos se pueden capturar diferentes modelos de computación los cuales se implementan en un conjunto de canales SystemC, implementados por F. Herrera y que pueden ver en sus trabajos [10] y [88], además de los ya mencionados [e)] y [mm)]. Estos canales implementan la semántica ejecutiva tales como redes de Kahn, diferentes escenarios de comunicación rendezvous, Synchronous Data Flow…

Relación UML/MARTE-ForSyDe/SystemC

En el trabajo [o)] se desarrolló una metodología que permite la especificación de sistemas "Synchronous Data Flow" (SDF), capturando todas las características que definen este MoC mediante modelos UML/MARTE, basándose en los trabajos [e)] y [f)] y en la línea de formalización en *Formalización de UML/MARTE: ForSyDe* del capítulo *Modelado de Sistemas Electrónicos*.

Considerando otra vez el mismo formalismo, se genera un SystemC-ForSyDe [11] que permite una simulación funcional formal del sistema capturado desde modelos UML/MARTE para sistemas cuya semántica de comportamiento es SDF. De esta manera se puede analizar las propiedades del sistema, en términos de tasas datos producidos/consumidos y estructura concurrente del sistema. Además de esto, con este código ForSyDe-SystemC, se puede establecer un proceso de exploración analítica para diseños de sistemas predecibles en tiempo.

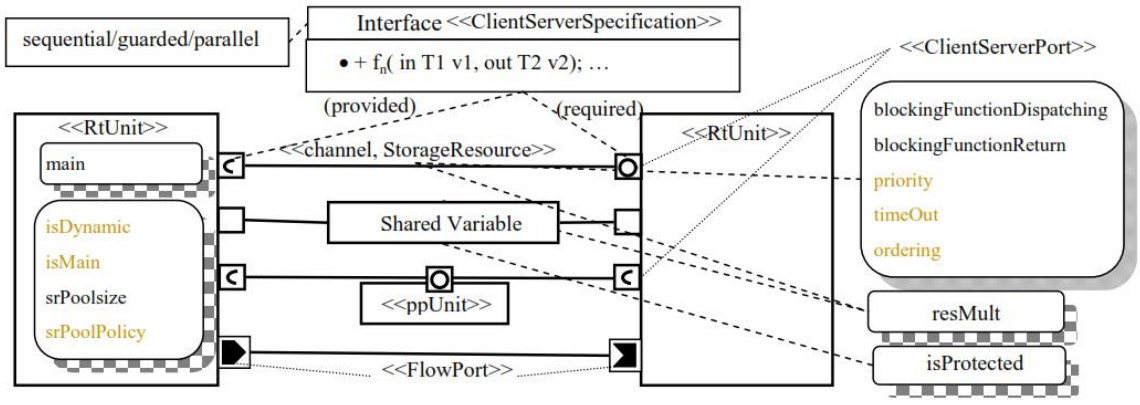


Figura 62 Esquema de la descripción de la aplicación y su semántica de ejecución

La Figura 62 (hecha por F. Herrera) muestra la información que es capturada en la aplicación, cuyos elementos están incluidos en las vistas de datos, funcional, de comunicación y de aplicación. Para ver qué significa cada uno de esto elementos de UML y MARTE, [oo)] y [uu)].

Para esto modelado, se ha de considerar una función de inicialización de los componentes de aplicación, no incluida en la descripción de esos elementos realizadas en la sección *Vistas: Modelo Independiente de Plataforma* del capítulo *Modelado de Sistemas Electrónicos*. La función de inicialización se muestra en la Figura 63. Esta función denota los valores por defecto de la función principal del componente y que se han de tener en cuenta cuando el componente se construya.

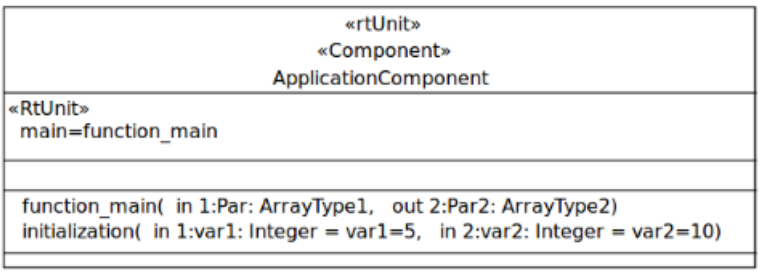


Figura 63 Componente de aplicación con su función de inicialización

La metodología de modelado UML/MARTE de esta tesis presenta una gran variedad expresiva de acuerdo a la alta variedad de atributos, semánticas de comunicación y alternativas de modelado, en lo que a la aplicación se refiere, como se ha ido mostrando en las secciones anteriores. El inconveniente es la consistencia del modelo y el cumplimiento de ciertas propiedades, como el determinismo o la ausencia de bloqueos, que no pueden ser garantizados en lo que a la aplicación se refiere; de ahí el papel que juega os formalismos matemáticos. Para ello, y en el contexto de los sistemas SDF, se van definir un conjunto de patrones de diseño, de los cuales se puede inferir un sistema SDF con todas sus propiedades a partir de los descrito en la sección *Canales y Concurrencia* del capítulo *Modelado de Sistemas Electrónicos.*

Patrón 1: Flujo de datos Explícito

El primer patrón se muestra en la Figura 64, donde se captura de forma explícita los flujos de datos como mecanismo de comunicación. Esto se hace mediante la utilización de "FlowPorts" en los cuales se denota el tipo de dato, y el número de datos que se esperan recibir en cada puerto (notación en la figura 1:p1:T1[N], donde T1 es el tipo de dato y [N] el número de datos que se requiere de esta entrada) para disparar la ejecución de la función asociada al componente f_A. La semántica SDF especifica que para disparar la función se han de tener suficientes datos en todas las entradas. Esto se denota asociando una condición de "join" a los puertos del componente de aplicación (Figura 64, hecha por F. Herrera).

De la misma forma, también se denota la cantidad de datos generados y su tipo asociado, especificado en los puertos de salida del componente de aplicación.

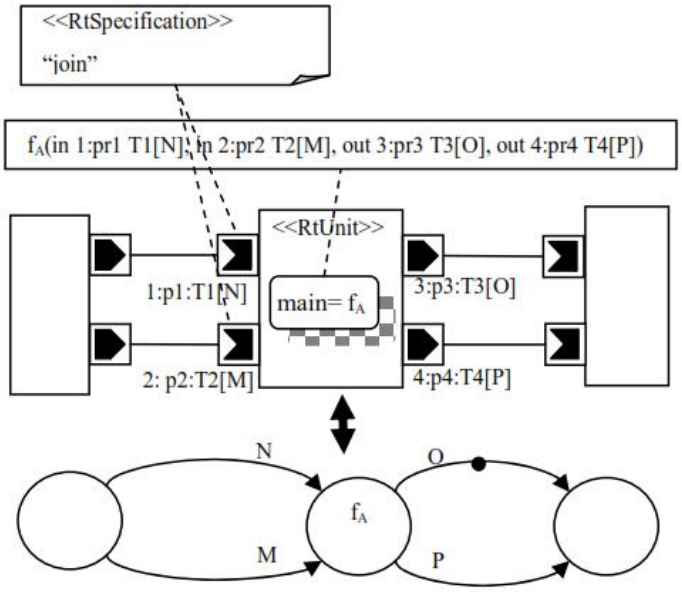


Figura 64 SDF: Patrón 1.

La metodología también permite la integración en un mismo componente de comunicación orientada a flujo de datos y a comunicación basada en llamadas a función, con una semántica de interfaz específica interfaz (sección *Canales y Concurrencia* del capítulo *Modelado de Sistemas Electrónicos*). Esta integración se hace en el componente "Transactor". Este componente presenta los dos tipos de puertos, como se puede ver en la Figura 65 (hecha por F. Herrera).

La Figura 65 a) captura una llamada a un servicio que requiere de algún dato y nunca retorna. La Figura 65 c) es similar al anterior, pero requiere procesar y retornar datos desde la parte de flujo de datos. La Figura 65 b) refleja la generación de una llamada a un servicio con datos provenientes de la parta de flujo de datos. La llamada es realizada una vez que todos los datos están disponibles en las correspondientes entradas y están disponibles para su consumo. La Figura 65 d) devuelve datos por la parte de flujo de datos. Tanto la Figura 65 a) como la Figura 65 b) se infieren nodos "Source" y "Sink" de SDF, respectivamente. Un nodo "Sink" es aquel únicamente requieren datos. Los nodos "Source" son aquellos que solo presentan datos de salida.

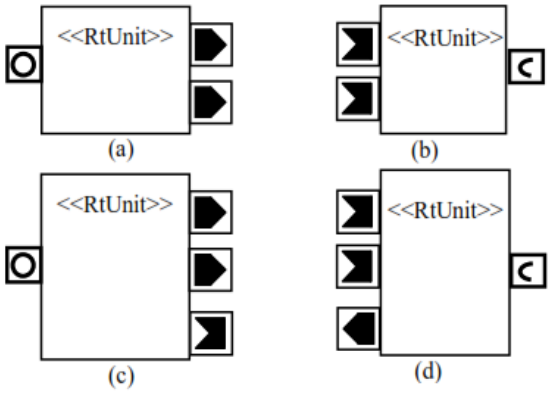


Figura 65 Componente "Transactor".

Patrón 2: Llamadas a función

Este patrón únicamente considera como mecanismo de comunicación llamadas a servicios de interfaz. Este patrón se muestra en la Figura 66 (hecha por F. Herrera) de forma simplificada. Los componentes únicamente presentan un puerto donde se ofrezca un interfaz. Ese interfaz únicamente tiene un servicio que corresponde con la función principal del componente, f_A. Esta función tienen varios argumentos (a, b y c) especificados con los correspondientes tipos de datos (T1, T2 y T3), declarados en la vista de datos.

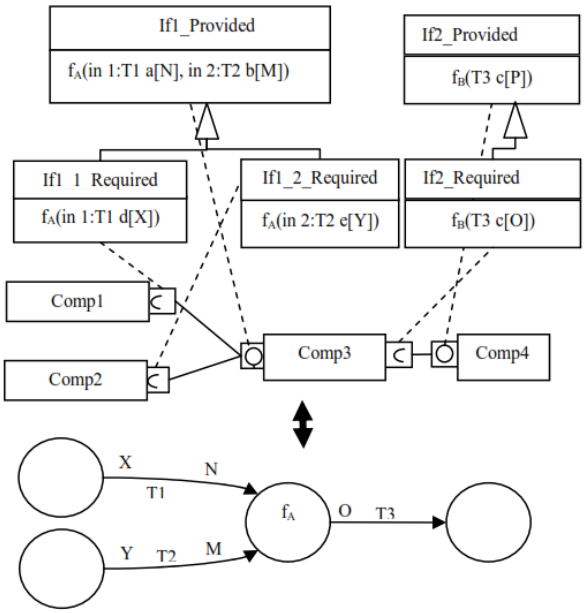


Figura 66 SDF: Patrón 2.

El patrón muestra la necesidad de presentar la asociación de un interfaz entre los puertos conectados. En un caso simple, un único puerto que proporciona un interfaz está conectado a un puerto que requiere dicho interfaz (en la Figura 66, los componentes Comp3-Comp4). Está asociación solo requiere que los tipos de datos de los parámetros (T3 en ambos casos) sean el mismo y que el servicio de los interfaces sea el mismo (f_B).

Cuando los tipos de datos son arrays o colecciones, la asociación entre las dimensiones no es necesaria.

Este patrón de modelado ha de hacer explícito la compatibilidad entre los interfaces; esto se hace mediante la relación de herencia entre dichas interfaces (Figura 66, parte superior de la misma); la interfaz requerida ha de heredarse de la interfaz proporcionada cundo se declaran dichas interfaces (en la vista funcional).

La Figura 66 también refleja la partición en los interfaces. Esta propiedad definida en la metodología permite dividir la llamada al servicio proporcionado en dos o más componentes en sus puertos requeridos (como fue descrito en la sección *Canales y Concurrencia* del capítulo *Modelado de Sistemas Electrónicos*). Estos puertos requeridos tienen que tener asociados interfaces que sean el resultado de la partición del interfaz proporcionado. De nuevo, la herencia de interfaces es usada para capturar de forma explícita este tipo de interconexión.

Patrón 3: Interfaces para transferencia de data

Un tercer tipo propuesto por E. Villar puede ser explicado como una variante del patrón tipo 1, tal que cada flujo de datos es transformado en una comunicación basada en llamadas a función, donde los interfaces tienen servicios del tipo “write” o “transfer” (Figura 67, hecha por F. Herrera) para denotar la transmisión de los datos. En este caso, las reglas de modelado son las mismas que en el patrón 2.

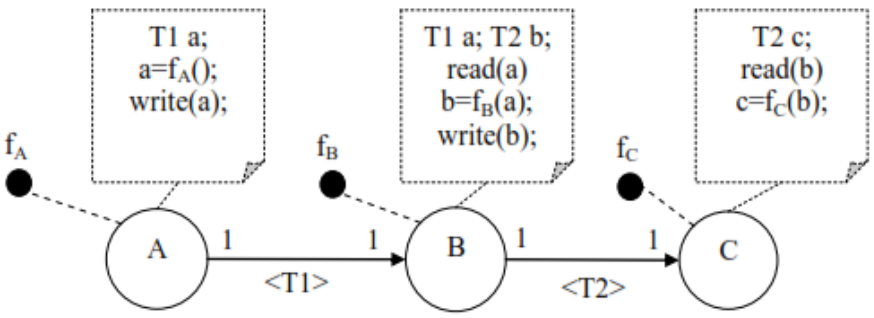


Figura 67 SDF: Patrón 3.

Para más detalles de toda la metodología de modelado ver [uu)].

Generación de código para simulación funcional

En este contexto, se desarrolló un conjunto de herramientas que permitiera la obtención del modelo de UML/MARTE a partir de una descripción del mismo, leyendo archivos XML que incluyera una descripción ForSyDe del sistema SDF (flecha ForSyDe2MARTE de la Figura 68, figura hecha por F. Herrera). También se desarrolló una generador de código que desde el modelo UML/MARTE, generase código SystemC/ForSyDe (flecha MARTE2ForSyDe de la Figura 68). Las herramientas, integradas en Eclipse, permiten la compilación del código SystemC/ForSyDe generado, su simulación, así como la generación de los archivos XML-ForSyDe que capturan el sistema. Esto permite la validación y simulación, enlazando con un entorno de DSE analítico basado en ForSyDe para el diseño de sistemas predecibles temporalmente.

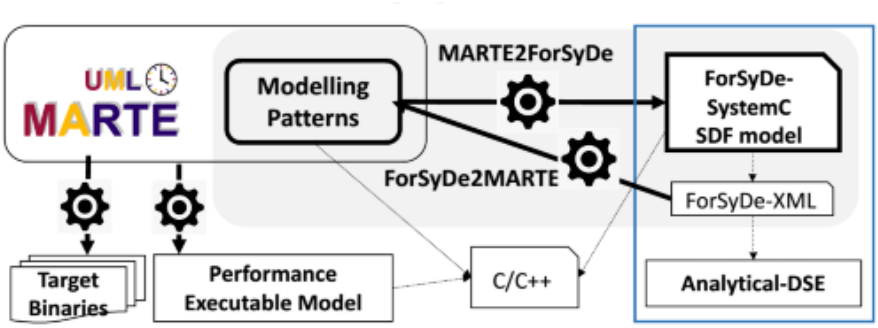


Figura 68 Entorno de herramientas

Resultados

Hasta ahora, los trabajos realizados permiten definir la aplicación en base a unos principios formales que le dan consistencia en términos de determinismo y propiedades de comportamiento bien definidas. El hecho de poder abstraer un modelo formal de dichos modelos permite establecer un proceso de transformación desde el modelo de alto nivel a una especificación ejecutable con las mismas propiedades semánticas (concurrencia y comunicación), estableciendo una simulación formal de dicha aplicación. De esta forma, mediante la utilización de HetSC y de SystemC-ForSyDe, dicha simulación funcional formal puede ser llevada a cabo, obteniendo resultados que dan información para valorar el diseño. Esto permite refinar la funcionalidad de acuerdo a los resultados obtenidos, haciendo un primer filtrado de todos los posibles diseños de la funcionalidad a implementar.

El trabajo UML/MARTE-SystemC/HetSC sentó las bases para desarrollar posteriores trabajos; la separación entre comunicación y cómputo es el principio en el cual se fundamenta todo el trabajo desarrollado en esta tesis en lo que especificación de la funcionalidad se refiere. El hecho de diferenciar elementos funcionales y de comunicación simplifica el estudio y el análisis de los sistemas concurrentes ya que las propiedades formales de cada uno de los elementos que conforman el sistema pueden ser obtenidas de manera más fácil. Otro aspecto que aportó esta metodología fue el hecho de usar componentes como elementos básicos para la creación del sistema, como forma de aglutinar y organizar la funcionalidad. También este trabajo sirvió para detectar carencias expresivas que tiene MARTE en lo que a capturar propiedades lógicas (bloqueos) de los mecanismos de comunicación. Esto fue de gran ayuda en posteriores trabajos a la hora de modelar canales de comunicación que obligó a la creación de nuevos mecanismos expresivos que solucionarán esta carencia de MARTE.

La labor realizada por mí en estos trabajos fue la realización de toda la metodología de modelados, así como de la implementación del conjunto de generadores asociados.

3.2 Modelo funcional con tiempos

En ocasiones, la simulación funcional no es posible sin cierta consideración de tiempos. En estos casos, se requiere de más información para validar el diseño de la aplicación y para obtener su implementación final. Por ello se ha de añadir información relativa a la plataforma donde se va a ejecutar dicha aplicación: en qué tipo de recursos se van a mapear los diferentes componentes de aplicación.

Para desarrollar la aplicación real, la noción de tiempo ha de incluirse desde las primeras etapas del proceso de diseño. Además de esto, como anteriormente mencionado, el hecho de que se verifique que la aplicación sea funcionalmente correcta no implica que el sistema esté bien diseñado. Hay que considerar el resto del sistema, la plataforma HW/SW y el mapeo sobre los recursos de esta plataforma. Por tanto, la aplicación se ha de refinar introduciendo nuevas propiedades que lo acerquen más a su implementación real.

Siguiendo en esta línea, se estableció una colaboración con el departamento ARCO [18] de la Universidad de Castilla-La Mancha dentro del proyecto DREAMS. Esta colaboración consistía el desarrollo de flujo completo que, tomando como base modelos de UML/MARTE se permitiera la síntesis de HW pero realizando una fase previa de simulación de cada diseño realizado. A partir de este modelo se obtiene una

especificación ejecutable en SystemC. Esta especificación incluye la distinción entre mapeos a recursos SW y HW, lo que permite estudiar el impacto que tiene el mapear un componente a un tipo u otro de recurso. Estos trabajos ([d)] y [t)]) se abordarán en más detalle en la sección *Generación de código: Síntesis* de *HW* de este.

Proyectos en los que este trabajo se desarrolló:

1. *SATURN, l) y [32]. Director principal: E. Villar; presupuesto: 193200€; periodo: 2008-2010.*
2. *CONTREX, [h)], [ll)], [ww)]. Director principal: E. Villar; presupuesto: 391336; periodo: 2013-2016.*
3. *DREAMS, [zz)]. Director principal: P. Sánchez; presupuesto: 141086€; periodo: 2011-2013.*

Publicaciones resultantes:

1. *Generating Heterogeneous Executable Specifications in SystemC from UML/MARTE Models, [e)].*
2. *Formal Foundations for the Generation of Heterogeneous Executable Specifications in SystemC from UML/MARTE Models, [i)].*
3. *Formal Support for Untimed MARTE-SystemC Interoperability, [j)].*
4. *Formal Modeling for UML/MARTE Concurrency Resources, [m)].*
5. *Formal Foundations for MARTE-SystemC Interoperability, [n)].*
6. *Enhancing Analyzability and Time Predictability in UML/MARTE Component-based Application Models, [o)].*
7. *Synthesis of Simulation and Implementation Code for OPENMAX Multimedia Heterogeneous Systems from UML/MARTE models, [d)].*
8. *Building a Dynamically Reconfigurable System through a High-Level Development Flow, [t)].*
9. *UML/MARTE methodology for automatic SystemC code generation of OPENMAX multimedia applications, [bb)].*
10. *Towards SystemC Code Generation from UML/MARTE Concurrent System-Level Models, [jj)]*
11. *Executable SystemC specification of the MARTE generic concurrent and communication resources under different Models of Computation, [kk)].*

4. Generación de código para la Síntesis de SW y la obtención de Ejecutables

Como se explicó en la sección *Información asociada a la síntesis* del capítulo *Metodología de Modelo Único*, la síntesis de SW ha sido utilizada para tres grandes áreas, ambas interrelacionadas y no ortogonales: canales y concurrencia, síntesis de SW para un recurso HW específico y síntesis de SW usando diferentes tipos de librerías de comunicaciones y servicios de sistema operativo.

La síntesis de SW es realizada por la herramienta eSSYN. Para que esta herramienta pueda generar la batería de archivos de código auxiliares que implementan las comunicaciones entre componentes y la estructura concurrente, dicha información capturada ha de ser transmitida a eSSYN. Esta trasmisión de información es realiza mediante un conjunto de archivos XML donde se anota toda la información del modelo, y que se generan mediante unos generadores de código implementados con Acceleo.

Una vez creado el modelo y leída la información del mismo en los XML, se inicia el proceso de síntesis de SW para obtener el correspondiente ejecutable para la plataforma de destino. Con esta información, se sintetizan el código necesario para iniciar los componentes de aplicación, lanzar los hilos estáticos, y se producen los canales con la semántica asociada. Para este propósito, el generador de código crea implementaciones ad-hoc para cada tipo de comunicación incluida en el modelo. Estas implementaciones incluyen las llamadas requeridas a los servicios de plataforma utilizados para proporcionar la estructura concurrente y de la comunicación (sección *Especificación del sistema: plataforma y heterogeneidad* del capítulo *Modelado de Sistemas Electrónicos*). Los archivos se crean ad hoc, en para minimizar la sobrecarga resultante. Esto significa que el código C creado para implementar aplicaciones, ejecuciones y comunicaciones se genera estrictamente dependiendo de la semántica del canal, del mapeo arquitectural, de los argumentos de los servicios, tamaños de datos, etc., especificados en el Modelo UML, en lugar de basarse en códigos generales que se deberían adaptar a las características del sistema bajo diseño.

Desde el modelo UML/MARTE, aparte de los archivos XML, se generan automática el conjunto de “makefile” necesarios para realizar la compilación.

Finalmente, código generado automáticamente y código funcional proporcionados por el usuario se compilan juntos, obteniéndose un ejecutable por cada especio de memoria presente en el modelo.

La estructura de forma de generar el código SW a partir de la información del modelo se describe en más detalle en [f)] y en [v)].

En esta línea de trabajo, la labor realizada por mí fue el desarrollo de la metodología de modelado completa (que se muestra en [c)] y en [v)], detallada en el manual [oo)]), la cual permitiera capturar en un modelo UML/MARTE todos los aspectos necesarios para realizar la síntesis: cosas tales como librerías, compiladores, opciones de compilación y enlazado APIs (sección *Información asociada a la síntesis* capítulo *Modelado de Sistemas Electrónicos*)...

También implementé el conjunto de generadores encargados de transformar la información capturada en el modelo en archivos XML. Estos archivos son la conexión entre el modelo y eSSYN. Esta herramienta, a partir de dichos archivos XML, hace la síntesis del código SW que va a ser llevado a la plataforma destino. Este código incluye toda la infraestructura que permite la comunicación entre los componentes de aplicación, con la semántica capturada en los canales, así como de toda la estructura concurrente especificada en el modelo. Todo esto es generado en función de la API seleccionada en el modelo para sintetizar el código y en función del compilador y opciones de compilación y enlazado (Figura 32).

A esto hay que añadir lo requerido para generar el código óptimo para los recursos HW específicos presentes en la plataforma, como compiladores específicos y condiciones de compilación y enlazado.

Finalmente, me encargué de la generación de los "makefiles" necesarios para compilar todo el código producido por eSSYN junto con el código funcional de cada componente (en función de a qué recurso de plataforma estuviera mapeado dicho componente de aplicación).

Este trabajo se realizó en colaboración con A. Nicolás y H. Posadas.

Proyectos en los que este trabajo se desarrolló:

1. *PHARAON, [g)] y [aaa)]. Director principal: E. Villar; presupuesto: 354000€; periodo: 2011-2014.*

Publicaciones resultantes:

1. *Automatic synthesis of communication and concurrency for exploring component-based system implementations considering UML channel semantics, [f)].*
2. *Automatic synthesis of embedded SW for evaluating physical implementation alternatives from UML/MARTE models supporting memory space separation, [c)].*
3. *Automatic Synthesis over multiple APIs from UML/MARTE Models for easy Platform Mapping and Reuse, [v)].*

5. Generación de código: Síntesis de HW

Como se mencionó en la sección *Modelo funcional con tiempos* de la sección *Análisis: Simulación funcional* de este capítulo, el trabajo de establecer un proceso de síntesis de HW surgió de la colaboración con el departamento ARCO de la Universidad de Castilla-La Mancha dentro del proyecto DREAMS.

Este trabajo consistió en desarrollar un flujo de diseño (Figura 69, realizada por D. de la Fuente) que permita realizar de forma semiautomática el mapeo HW de componentes de aplicación, incluyendo la exploración de distintas alternativas de implementación que permitan optimizar el sistema final. Este flujo de diseño incluye una fase de simulación del sistema diseñado para su verificación. Partiendo del modelo UML/MARTE, se genera una especificación ejecutable SystemC que incluye representación de cada componente con un conjunto de propiedades que los caracterizan y sus conexiones asociadas. De esta forma, se puede simular el sistema y estudiar sus prestaciones, realizando un proceso semiautomático de análisis de prestaciones del diseño. Acorde a este proceso de análisis, se decide la mejor configuración de sistema, generando la correspondiente infraestructura VHDL para implementar la síntesis HW sobre una FPGA.

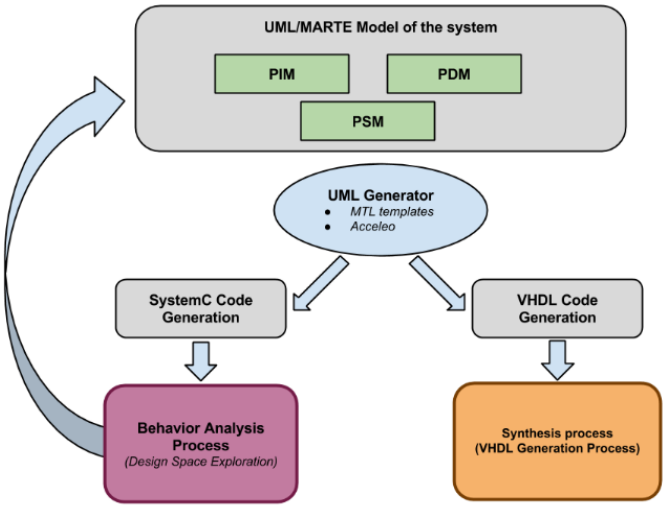


Figura 69 Flujo para la síntesis de HW

5.1 OpenMAX

Tomando como base el flujo anteriormente descrito, esta colaboración desarrollo dos líneas de trabajo. La primera, consiste en el desarrollo de sistemas empotrados multimedia basados en el estándar OpenMAX [17].

La metodología de modelado UML/MARTE se extendió para permitir el modelado de los sistemas multimedia basados en OpenMAX, añadiendo nuevos mecanismos de comunicación, así como poder especificar el conjunto de propiedades de un componente OpenMAX potencialmente explorables durante las simulaciones SystemC. Todas propiedades de los componentes OpenMAX fueron descritas en *Sistemas OpenMAX* de la sección *Información asociada a la síntesis* del capítulo *Modelado de Sistemas Electrónicos*.

Otro aspecto que se puede explorar (mediante la simulación SystemC) es la caracterización de cada componente OpenMAX como HW o SW. En este caso, la agrupación de los componentes de aplicación en espacios de memoria no era necesaria, ya que la implementación HW/SW del componente ya viene definida por el propio estándar.

Para establecer un proceso semiautomático de exploración del diseño, la asignación de un componente OpenMAX como HW o SW puede ser analizada mediante las simulaciones SystemC. Además de esto, se pueden estudiar y ajustar los valores de las propiedades que caracterizan al componente OpenMAX.

Una vez realizadas las sucesivas simulaciones y seleccionada la deseada, se genera el código VHDL asociado a cada componente que permite su implementación en la FPGA. Todo esto se presentó en el trabajo [d)].

Se mostrará un ejemplo en la sección *Síntesis de HW: OpenMAX* del capítulo *Ejemplos de Uso y Resultados*.

El trabajo realizado como parte de la tesis fue la ampliación de la metodología para poder modelar un componente OpenMAX (Figura 33), así como añadir nuevos tipos de canales (Figura 34) que tuvieran las propiedades de los sistemas basados en OpenMAX. De nuevo toda la infraestructura de generación de código fue implementada por mí.

5.2 Reconfigurabilidad

La segunda línea de trabajo que se realizó con este grupo consiste en permitir el desarrollo de sistemas parcialmente reconfigurables; un diseñador puede modificar parte de la configuración de un recurso de la plataforma en tiempo de ejecución y, por lo tanto,

el diseño inicial podrá ser modificado después de su despliegue sin necesidad de una reconfiguración completa.

Todo es nuevo flujo se describe en el artículo [t)]. Aquí, de nuevo se usa SystemC para simular el sistema y hacer un análisis previo de sus prestaciones. Una vez que se ha acometido esta tarea, se selecciona aquella configuración del sistema que haya tenido unas mejores prestaciones y se genera su correspondiente implementación HW. Esta generación automática crea toda la infraestructura necesaria para permitir la reconfiguración. En ella destaca el "reconfiguation engine", elemento que ofrece los servicios para permitir la reconfiguración dinámica [t)], conformando la zona estática de la FPGA. De la misma forma, y también desde el modelo, se generan la zona dinámica reconfigurable de la FPGA. Para cada área reconfigurable, se define una ubicación geométrica, los recursos asignados y un nombre. De nuevo, toda esta información se ha de especificar en el modelo UML/MARTE (Figura 70, realizada por D. de la Fuente).

Para un mejor mapeo en la plataforma, donde se iban a tener elementos heterogéneos como CPUs y FPGA, los espacios de memoria son necesarios en estos sistemas. Por tanto, era necesario poder agrupar aquellos componentes de aplicación cuyo mapeo se iba a realizar en cada recuro HW.

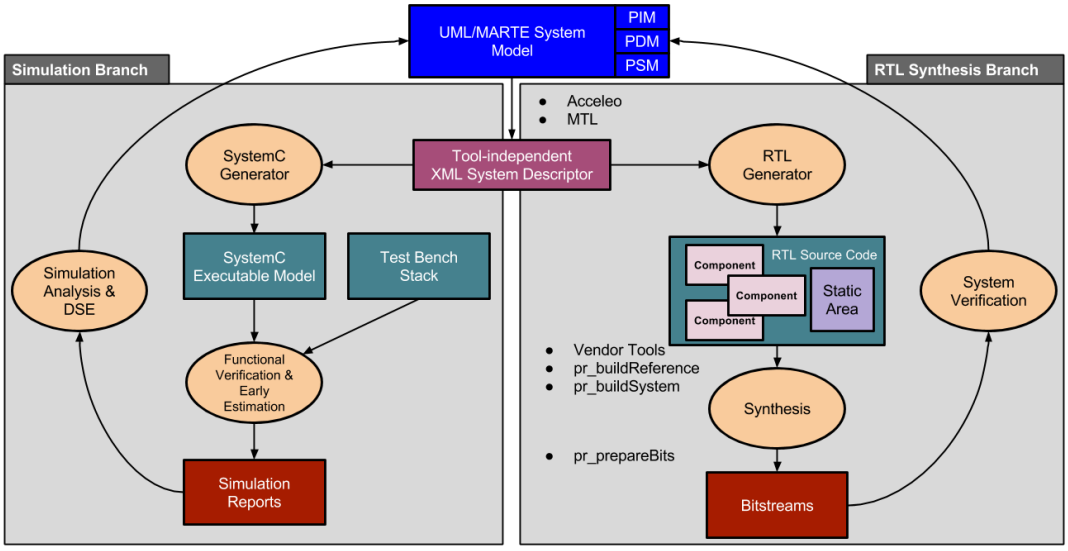


Figura 70 Diagrama de flujo para sistemas reconfigurables

La labor realizada en el ámbito de la tesis consistió en el desarrollo de la metodología de modelado de UML/MARTE, tomando como base la descrita en [c)] y [d)], para poder capturar toda la información necesaria para caracterizar la plataforma (y

en concreto la FPGA, Figura 71) con el fin de automatizar la generación de toda la estructura de archivos que implementase el "reconfiguation engine". También, de nuevo, se implementó el correspondiente generador de SystemC para poder realizar la simulación del sistema.

En este flujo, la mayor aportación de la metodología es el hecho de añadir como recurso de la plataforma la FPGA y desarrollar el proceso de implementación. Como se puede ver en la Figura 71, también el modelo de FPGA lleva asociado un conjunto de atributos que son usados para poder generar los correspondientes archivos para el "toolkit" de Xilinx. Con esto, y usando de nuevo los espacios de memoria, se pueden mapear los correspondientes componentes de aplicación a dicha FPGA o a otros recursos de procesado de la plataforma.

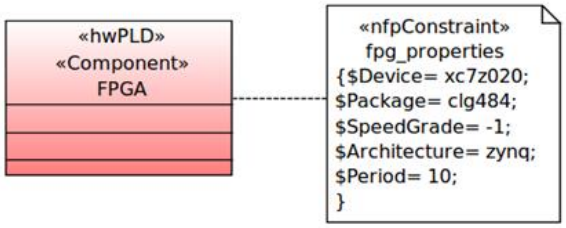


Figura 71 Modelado de FPGA.

Proyectos en los que este trabajo se desarrolló:

1. *DREAMS, [zz)]. Director principal: P. Sánchez; presupuesto: 141086€; periodo: 2011-2013.*

Publicaciones resultantes:

1. *Synthesis of Simulation and Implementation Code for OPENMAX Multimedia Heterogeneous Systems from UML/MARTE models, [d)].*
2. *UML/MARTE methodology for automatic SystemC code generation of OPENMAX multimedia applications, [bb)].*
3. *Building a Dynamically Reconfigurable System through a High-Level Development Flow, [t)].*

6. Análisis: Análisis de Prestaciones

Con ese tipo de análisis, el diseñador puede estudiar e rendimiento de sistema en función de un conjunto de métricas no funcionales, como consumo de potencia, latencia… ([qq)]). También se puede poner restricciones a dichas métricas y comprobar que el sistema las cumple o no (Figura 72).

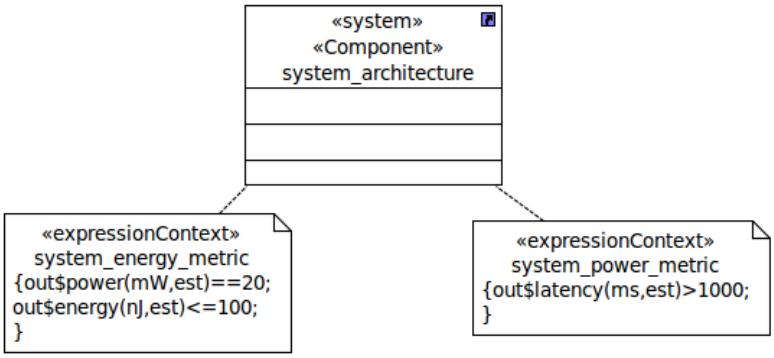


Figura 72 Métricas del Sistema

Además de considerar la aplicación, en este tipo de análisis se incluye la plataforma donde la aplicación se va a ejecutar. Este hecho introduce elementos que pueden afectar al rendimiento del sistema y que se han de considerar durante el proceso de diseño. Parámetros de la plataforma como frecuencia de los procesadores, tamaño de memorias (por ejemplo, de caches), tiempo de acceso a memorias, son ejemplos que tienen impacto en el rendimiento del sistema y, por tanto, que se deben considerar.

En este tipo de análisis también es posible obtener métricas asociadas a elementos de la plataforma. Ejemplos de estas métricas son los mises de una caché, consumo de un procesador… ([qq)]). De esta forma se puede analizar de una manera más específica el rendimiento de un elemento HW concreto.

Para ir introduciendo estos aspectos y ver su impacto en los procesos de diseño se utilizan simuladores de modelos virtuales. En el simulador se describe una plataforma virtual, donde cada tipo de componente HW de la misma es caracterizado por un conjunto de propiedades ([oo)]). En esta plataforma virtual se ejecuta la aplicación generada mediante el proceso de síntesis de SW descrito en la sección *Generación de código* de este capítulo, de forma que la herramienta de simulación va obteniendo de manera automática los tiempos y otros parámetros no funcionales resultantes de ejecutar ese código en esa plataforma.

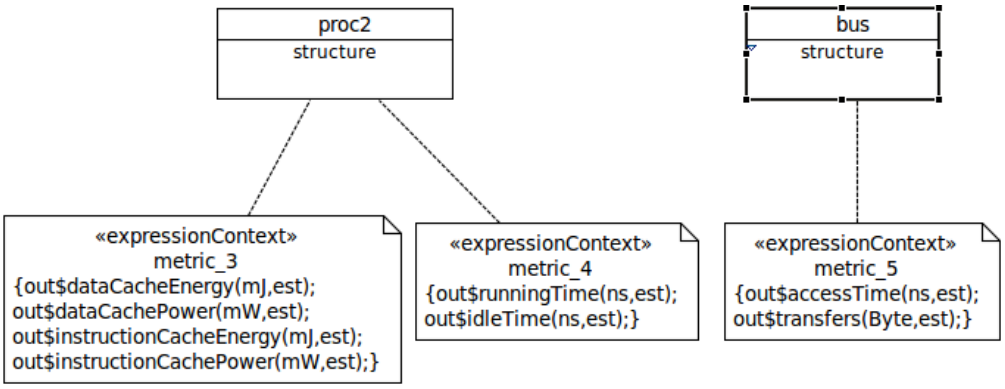


Figura 73 Métricas de componentes HW

En el contexto de este trabajo, se ha aplicado un simulador, utilizando dos versiones diferentes, que, partiendo de la información capturada en el modelo, obtiene métricas de rendimiento del sistema completo. En el trabajo [p)], la herramienta de simulación usada es SCoPE ([19], [20]) y en [q)] se ha usado una nueva versión de esta herramienta, que se llama VIPPE [21] (las cuales se presentaron en la sección *Otros Lenguajes y Herramientas Usados en la Tesis* del capítulo *Estado del Arte*).

Todo el proceso de análisis es automático; en el modelo se especifican las propiedades de cada componente HW de la plataforma ([qq)]), los mapeos arquitecturales y capturan los diferentes escenarios de análisis y las diferentes métricas del sistema que se quieren analizar. Una vez hecho esto, desde el mismo entorno de desarrollo se transfiere la información del modelo al simulador. Con dicha información, el simular construye un modelo virtual de la plataforma, en función de las características capturadas en el modelo. Una vez hecho esto, se crea toda la infraestructura necesaria para ejecutar el SW de aplicación en dicho modelo de plataforma. De esta forma, se obtienen las métricas de rendimiento del sistema consideradas en el análisis. Estos simuladores generan un conjunto de estimaciones del sistema que permiten sacar conclusiones sobre el impacto de la plataforma en el rendimiento de la aplicación diseñada, permitiendo un proceso de refinamiento manual o automático (sección *Análisis: Exploración del Espacio de Diseño* de este capítulo).

La transmisión de la información a los simuladores se hace a través de archivos XML que incluyen toda la información del sistema necesaria para realizar la simulación, principalmente información de la plataforma y del mapeo de la aplicación en dicha plataforma. A esto hay que añadir qué tipo de métricas se desea que el simulador genere.

Para integrar el simulador dentro del entorno se necesita realizar, principalmente, dos tareas; la generación de los XML y la lectura de dichos XML por parte del simulador. En ambos trabajos [p)] y [q)] implementé el generador de XML. Sin embargo, únicamente en el caso del entorno con el simulador de VIPPE realicé el lector de XML. Dicho lector de XML se implementó en C, usando la librería libxml2.

Además de esto, es necesario el SW de aplicación, el cual es generado a través del proceso de síntesis de SW descrito en la sección *Generación de código* de este capítulo.

En esta misma línea se puede estudiar el impacto sobre el rendimiento de la estructura concurrente y los canales de comunicación. Aunque esto se mostrará en más detalle en la sección *Canales y Concurrencia* del capítulo *Ejemplos de Uso*, se puede ya decir que mediante un proceso de síntesis de SW, el diseñador puede ver las prestaciones obtenidas de su aplicación, sobre una plataforma u plataforma dadas. De esta forma se podrá seleccionar mejor la configuración final del sistema.

Todos los detalles de la metodología en [qq)].

Este trabajo se ha hecho con la colaboración de H. Posadas, L. Díaz y F. Herrera.

Proyectos en los que este trabajo se desarrolló:

1. *COMPLEX, [32], [xx)]. Director principal: E. Villar; presupuesto: 325588€; periodo: 2009-2013.*
2. *PHARAON, [g)] y [aaa)]. Director principal: E. Villar; presupuesto: 354000€; periodo: 2011-2014.*
3. *CONTREX, [h)], [ll)], [ww)]. Director principal: E. Villar; presupuesto: 391336; periodo: 2013-2016.*

Publicaciones resultantes:

1. *The COMPLEX methodology for UML/MARTE modeling and design-space exploration of embedded systems, [b)].*
2. *UML/MARTE methodology for high-level system estimation and optimal synthesis, [p)].*
3. *System level design framework for many-core architectures, [q)].*
4. *A MDD Methodology for Specification of Embedded Systems and Automatic Generation of Fast Configurable and Executable Performance Models, [ee)].*

7. Análisis: Análisis de Planificabilidad

Otra de las actividades que se pueden dar durante el proceso de diseño es un análisis que, a partir de la estimación comportamiento temporal de los diferentes aspectos de la funcionalidad del sistema, permita analizar de forma estática si el diseño resultante será capaz de cumplir con todos los requerimientos del diseño. Este análisis es más relevante en aplicaciones de tiempo real "hard" donde la ejecución de la funcionalidad tiene requisitos temporales de estricto cumplimiento.

En estos análisis se estudian potenciales mapeos aplicación-plataforma en función de las propiedades de tiempo real de la aplicación y el tipo de recursos disponibles en la plataforma. De todos los potenciales mapeos que se pueden considerar, los análisis de planificabilidad nos muestran si se cumplen las restricciones temporales de cada elemento de la funcionalidad en relación a sus tiempos de ejecución o a sus plazos de ejecución máximos permitidos. De esta forma se puede saber si el mapeo realizado y los recursos de la plataforma considerados son suficientes o no para cumplir dichas restricciones temporales. De esta forma, se filtran todos los posibles mapeos eligiendo un subconjunto de ellos.

Para poder realizar este análisis hay que caracterizar la funcionalidad con los diferentes tiempos de ejecución de los servicios/métodos de los componentes de aplicación. Estos tiempos pueden ser obtenidos mediante simulaciones con VIPPE.

Además de esto, también es posible detectar carencias de los componentes HW de la plataforma, ya que se puede concluir que los recursos de plataforma considerados son insuficientes para los requerimientos temporales del sistema, teniendo que añadir unos nuevos o modificar los ya incluidos en el diseño.

Otro tipo de información que nos proporciona este análisis es la de poder estudiar la carga de utilización de los procesadores. En un sistema con criticidad mixta ([u)], [w)]), se establece una jerarquía entre las tareas a ejecutar, dependiendo del impacto en el sistema si una de ellas no es ejecutada en tiempo. De este tipo de análisis se obtienen estimaciones de la utilización de los diferentes recursos de procesado de la plataforma, permitiendo balancear el sistema. De esta forma, se tiene una medida de la disponibilidad de los recursos de procesado para ejecutar tareas menos críticas.

Para poder desarrollar modelos de análisis de planificabilidad, la metodología UML/MARTE de modelado se enriqueció con nuevos conceptos que permitan incluir en el modelo toda la información necesaria para permitir la ejecución de un análisis de

planificabilidad. Estos análisis de planificabilidad se posibilitaron mediante la integración de la herramienta MAST [24] y su metodología de modelado asociada.

Los conceptos añadidos a la metodología se describen en [s)], abarcando elementos que se incluyen en las vistas "ApplicationView", "ConcurrecyView" y de la "ArchitecturalView". Esta nueva información hace referencia a propiedades que son necesarias para hacer este tipo de análisis; propiedades como el peor y el mejor tiempo de ejecución de las funciones de los componentes de aplicación, o las prioridades de las tareas son añadidas al modelo.

Además de enriquecer las características de los elementos de las vistas anteriores, se definió una nueva vista, "SchedulabilityView", donde se especifican los modelos de análisis concretos, que incluyen la secuencia de funciones y los recursos utilizados para su ejecución. Para este fin, se utilizan diagramas de actividad, donde cada acción de dicho diagrama representa la ejecución de un conjunto de funciones, llevadas a cabo por una tarea específica.

Aparte de ampliar la metodología UML/MARTE presentada en este trabajo con todo lo necesario para crear de modelos de análisis de planificabilidad, lo siguiente fue el desarrollo de la herramienta que permitiera ese análisis. Aparte de la herramienta MAST, el mismo grupo de personas que la crearon, desarrollaron un entorno que permite lanzar MAST de forma automática desde modelos UML/MARTE, de acuerdo a su propia metodología de modelado ([6] y [7]). Tomando como base esto, ese entornó se modificó por mí de acuerdo a la metodología presentada en esta tesis: algunas modificaciones en el generador de código previamente implementado, adaptándolo a la metodología basada en las vistas del modelo y añadiendo todos los cambios necesarios para obtener la información necesaria de cada elemento contenido en cada vista [s)]. De esta forma MAST se integró en un entorno de desarrollo más completo, de amplio espectro de aplicación, donde las herramientas de dicho entorno proporcionan la información temporal necesaria para ejecución de la herramienta.

Una vez hechos estos cambios, se integró dicho entorno de análisis de planificabilidad en el entorno de desarrollo presentado en esta tesis.

Más detalles en [ss)].

Este trabajo se realizó colaborando con J. Medina.

Proyectos en los que este trabajo se desarrolló:

1. *CONTREX, [h)], [ll)], [ww)].*

Publicaciones resultantes:

1. *UML-Based Single-Source Approach for Evaluation and optimization of Mixed-Critical Embedded Systems, [s)].*

8. Análisis: Exploración del Espacio de Diseño

Gracias a las actividades de análisis anteriores es posible estudiar si una configuración del sistema cumple con los objetivos fijados o no. Sin embargo, en la búsqueda de soluciones válidas, rara vez se encuentra la solución más adecuada con la primera configuración elegida. En consecuencia, es habitual tener que resolver un proceso de exploración de las alternativas de diseño disponibles. Para resolver esta búsqueda de manera sencilla, la infraestructura desarrollada ha incluido una etapa de exploración.

Esta etapa del entorno de desarrollo permite explorar y analizar el conjunto de las variables de diseño que se pueden considerar sobre el sistema a implementar. Este estudio viene dado por los requisitos funcionales que se especifican sobre el sistema desde el inicio del diseño.

Este proceso de análisis y de evaluación de las prestaciones del sistema en función de un conjunto de variables de diseño se puede llevar a cabo de forma semiautomática o de forma automática.

8.1 Exploración semiautomática

La exploración semiautomática consiste en generar, de forma manual e individual, diferentes versiones de la especificación del sistema, las cuales son evaluadas en función del conjunto de métricas que se desea analizar (sección *Análisis: Análisis de Prestaciones* de este capítulo). El diseñador implementa un proceso de refinamiento del sistema por el cual, a luz de las métricas de rendimiento reportadas, él va haciendo cambios y ajustes en el modelo del sistema con el fin de cumplir sus requerimientos.

En el proceso semiautomático, el diseñador realiza un primer modelo del sistema completo, se evalúa esa versión de forma que se estudia el rendimiento del sistema y se obtienen las conclusiones pertinentes. Luego, el diseñador realiza cambios en el modelo (propiedades de los componentes HW, mapeo arquitectural, semántica de los canales…), capturando un nuevo escenario de evaluación en función de los cambios hechos en el modelo. De nuevo, haciendo uso de la misma infraestructura de herramientas, genera una nueva especificación del sistema y analiza las prestaciones obtenidas del mismo.

Este proceso se reitera las veces que se necesitan, generando una batería de versiones del sistema en función de los escenarios de análisis considerados. Sin embargo, este proceso es útil y manejable únicamente si el número de esos escenarios no es muy

elevado. De otra forma, esto puede resultar ser una tarea muy costosa en tiempo, que incluso puede llegar a ser inmanejable.

Exploración basada en simulaciones SystemC

En primer lugar, se puede englobar como exploración del espacio de diseño semiautomática el trabajo [d)], donde las simulaciones SystemC se usan para encontrar la mejor configuración de los componentes OpenMAX que conforman nuestro sistema. En dicho trabajo se usó como ejemplo un procesador de imagen SOBEL (ejemplo que se mostrará en más detalle en la sección *Síntesis de HW: OpenMAX* del capítulo *Ejemplos de Uso y Resultados*). Cada uno de sus componentes (Figura 33) fue caracterizado por un conjunto de propiedades. En sucesivas simulaciones SystemC se fueron estudiando cada uno de los valores considerados de dichas propiedades para cada componente. En cada una de ellas, los parámetros de configuración de cada componente (Figura 33) se van variando y se van observando cual es la mejor combinación de dichos valores. Los valores de estas propiedades son cambiadas manualmente en el modelo. Luego las especificaciones SystemC son automáticamente generadas, lo que hace que el proceso de exploración relativamente rápido, aunque no automático. Finalmente, con la mejor configuración para cada componente, se obtiene la implementación OpenMAX de cada uno de dichos componentes [d)].

También se puede incluir en esta sección todo el proceso de simulación y análisis del sistema de los sistemas reconfigurables mostrados en *Reconfigurabilidad* de la sección *Generación de código: Síntesis de HW* de este capítulo. Un ejemplo de esto se mostrará en la sección *Síntesis de HW: Reconfigurabilidad* del capítulo *Ejemplos de Uso y Resultados*.

Estos trabajos se realizaron en colaboración con el grupo ARCO de la Universidad de Castilla-La Mancha.

Proyectos en los que este trabajo se desarrolló:

1. *DREAMS, [zz)]. Director principal: P. Sánchez; presupuesto: 141086€; periodo: 2011-2013.*

Publicaciones resultantes:

1. *Synthesis of Simulation and Implementation Code for OPENMAX Multimedia Heterogeneous Systems from UML/MARTE models, [d)].*
2. *Building a Dynamically Reconfigurable System through a High-Level Development Flow, [t)].*

Exploración basada en simuladores temporales

También con los simuladores se puede implementar un proceso de exploración manual. Atendiendo a las métricas del sistema a estudiar, se va modificando el modelo y mediante la herramienta VIPPE (trabajos [p)] y [q)]), se van obteniendo las diferentes métricas a estudiar.

Un caso específico es la exploración semiautomática de diseño son las redes de sensores. Como se puede ver en [a)], una configuración de red puede ser estudiada bajo diferentes tipos de atacantes. En cada escenario de prueba, el modelo tiene que ser modificado de forma concreta para esa prueba; tantas pruebas se hacen tantas veces hay que modificar el modelo. Esto se mostrará en más detalle en la sección *Redes de Sensores* del capítulo *Ejemplos de Uso y Resultados.*

Con el fin de realizar la simulación de la red, toda la información de la misma es capturada en archivos XML, que son leídos por el simulador SCoPE, el cual ha sido modificado para poder simular redes ([a)] y [87]). Tanto el generador de archivos XML como la funcionalidad auxiliar para leer dichos archivos y transmitir la información capturada en esos archivos al simulador fueron realizados por mí.

Este trabajo se realizó colaborando con H. Posadas y L. Díaz.

Proyectos en los que este trabajo se desarrolló:

1. *COMPLEX, [32], [xx)]. Director principal: E. Villar; presupuesto: 325588€; periodo: 2009-2013.*
2. *PHARAON, [g)] y [aaa)]. Director principal: E. Villar; presupuesto: 354000€; periodo: 2011-2014.*
3. *CONTREX, [h)], [ll)], [ww)].*
4. *TOISE, [yy)]. Director principal: P. Sánchez; presupuesto: 163861€; periodo: 2011-2013.*

Publicaciones resultantes:

1. *High-Level design of wireless sensor networks form performance optimization under security hazards, [a)].*
2. *The COMPLEX methodology for UML/MARTE modeling and design-space exploration of embedded systems, [b)].*
3. *UML/MARTE methodology for high-level system estimation and optimal synthesis, [p)].*
4. *System level design framework for many-core architectures, [q)].*
5. *Modeling and Simulation of secure wireless sensor network, [r)].*

Exploración en la plataforma real

Por último, el proceso de síntesis de SW descrito en secciones anteriores puede servir para varios propósitos. En primero, y más obvio, es la implementación final de nuestra aplicación sobre la plataforma destino. En segundo lugar, la síntesis SW puede ser usada como mecanismo de exploración del diseño, implementando diferentes versiones de sistema en busca de la mejor de ellas.

Durante el proceso de diseño se puede tener implementadas diferentes versiones de código de la misma funcionalidad dependiendo del recurso HW en el que se quiera ejecutar dicho código. Estas versiones del código pueden responder a diferentes etapas del proceso de diseño en las que se parte de una implementación genérica del código, orientada a la ejecución en un procesador, y en sucesivas etapas de refinamiento, dicho código se modifica para poder ser mapeado a otros recursos HW, obtener un óptimo rendimiento de dicho recurso HW. Para que esto sea posible, la metodología de modelado permite asociar a cada componente de aplicación diferentes versiones de la misma funcionalidad. De esta forma, se puede tener documentada la evolución del desarrollo del sistema y se permite la utilización de más recursos HW de la plataforma de manera más óptima.

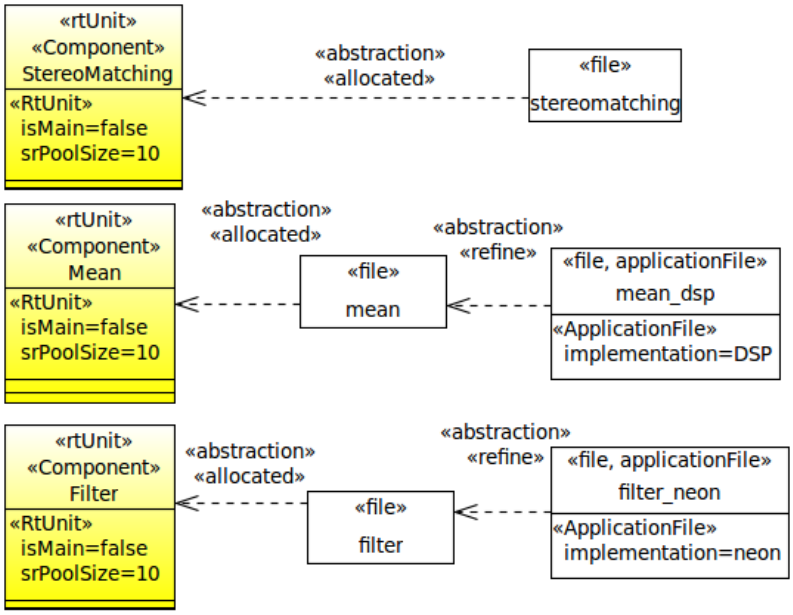


Figura 74 Refinamiento de archivos funcionales para un recurso HW específico

Como ejemplo, se muestra la Figura 74. En dicha imagen se puede ver una serie de componentes de aplicación (coloreados de amarillo) a los que se les ha asociado unos elementos que representan archivos con el correspondiente código funcional. En dos de ellos, estos se han refinado con otros archivos que representan el código optimizado para un determinado recurso HW (DSP y co-procesador NEON, respectivamente). De esta forma, y dependiendo del mapeo de ese componente, uno y otro archivo será añadido en los correspondientes "makefile" para su compilación de forma automática.

En la sección *Exploración de Recursos HW* del capítulo *Ejemplos de Uso y Resultados* se mostrará un ejemplo donde se aprecie dicho impacto.

Otro aspecto que se puede explorar durante el proceso diseño es cómo se implementan las comunicaciones entre los componentes de la aplicación; qué API y qué tipo de comunicación se usa. En la sección *Exploración de Recursos SW* del capítulo *Ejemplos de Uso y Resultados*, se muestran los resultados obtenidos con la infraestructura desarrollada en la tesis, de tal forma que dichos resultados indican cómo la API seleccionada para el experimento provoca un rendimiento diferente del sistema.

También se puede explorar en placa mediante la síntesis de SW el impacto de los canales y de la estructura concurrente derivada en base a lo descrito en la sección *Canales y Concurrencia* del capítulo *Modelado de Sistemas Electrónicos*. En la sección *Canales*

y Concurrencia del capítulo *Ejemplos de Uso y Resultados* se mostrarán un ejemplo donde se aprecie dicho impacto.

Este trabajo se realizó colaborando con A. Nicolás y H. Posadas.

Proyectos en los que este trabajo se desarrolló:

1. *PHARAON, [g)] y [aaa)]. Director principal: E. Villar; presupuesto: 354000€; periodo: 2011-2014.*

Publicaciones resultantes:

1. *Automatic synthesis of communication and concurrency for exploring component-based system implementations considering UML channel semantics, [f)].*
2. *Automatic synthesis of embedded SW for evaluating physical implementation alternatives from UML/MARTE models supporting memory space separation, [c)].*
3. *Automatic Synthesis over multiple APIs from UML/MARTE Models for easy Platform Mapping and Reuse, [v)].*
4. *Automatic Concurrency generation through Communication Data Splitting based on UML-MARTE Models, [y)].*
5. *Automatic synthesis of Embedded SW Communications from UML/MARTE models supporting memory-space separation, [dd)].*

8.2 Exploración Automática

El proceso manual anteriormente presentado puede ser muy costoso. En un proceso de análisis se puede querer considerar todas las potenciales alternativas que pueden abarcar tanto la propia arquitectura de la plataforma HW/SW como el mapeo arquitectural. Además de esto, se pueden considerar otros parámetros que tienen relación con las características de los recursos de la plataforma tales como tamaños de caché, tamaños de memoria, frecuencia de procesadores, anchos de banda de los buses, etc. Por tanto, las múltiples configuraciones que se pueden dar, considerando todos estos potenciales valores hace que afrontarlo de forma manual sea inabordable, necesitándose mecanismos automáticos para ello.

En [b)] se presenta un entorno completo que, basados en modelos UML/MARTE, permite establecer un proceso de exploración del espacio de diseño automático y

completo. Para ello, el modelo debe contener la especificaron de todas las variables de exploración, así como todos los potenciales valores asociadas a dichas variables. Para este fin se desarrolló un conjunto de nuevos conceptos de modelado que, junto con otros proporcionados por MARTE, permitían capturar todas estas variables de diseño, con sus valores asociados. Estos nuevos conceptos, así como la metodología de modelado se desarrollaron en colaboración con los autores de artículo [b)], siendo mi principal aportación todo lo referente al modelado de los recursos de la plataforma y sus propiedades, las reglas para usar esos nuevos conceptos de modelado y la forma de especificar las reglas de diseño para reducir el espacio de exploración considerado. Además de esto, todo el entorno de generación de código (archivos XML) que permitía automatizar el proceso exploración del espacio de diseño desde el modelo hacia las diferentes herramientas fue implementado por mí. En un principio, estas herramientas eran SCoPE y MOST. Este parte del entorno funcionaba de la siguiente manera; MOST leía un archivo XML que contenía todos los valores de cada variable de diseño del modelo. Seleccionaba un valor para cada variable de diseño y lanzaba una simulación de SCoPE. MOST recogía el valor de las métricas considerabas y volvía a iterar el proceso. Cuando finalizaba, proporcionaba un análisis de dichas métricas para las variables de diseño. SCoPE fue sustituido como simulador por VIPPE con posterioridad.

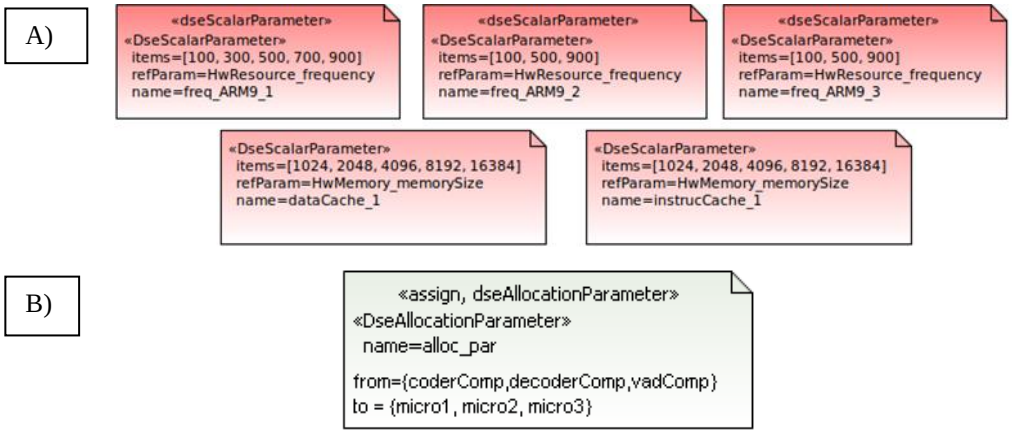


Figura 75 Especificación de variables de diseño

La Figura 75 muestra algunos ejemplos de cómo se capturaban estas variables de diseño a explorar. Para este fin se crearon nuevos elementos expresivos; en la Figura 75 los elementos “DseScalarParameter” y “DseAllocationParameter” se crearon ad-hoc para poder capturar dichas variables. Estos elementos expresivos fueron desarrollados por F. Ferrero y F. Herrera, con alguna pequeña contribución mía.

En la parte superior de la Figura 75, se muestran ejemplos de variables de exploración asociados propiedades de recursos HW (frecuencias, y tamaños de memoria). En la parte inferior de la figura, se muestran como capturar variables para explorar el mapeo de componentes de aplicación. En dicha imagen los componentes "coderComp", "decoderComp" y "vadComp" se pueden desplegar en los procesadores "micro1", "micro2" y "micro3" de todas las combinaciones posibles.

También era posible establecer reglas que restringieran posibles valores de las variables de diseño. La Figura 76 muestra "DseRule" que es el mecanismo expresivo que se creó para reducir los posibles valores del parámetro de exploración, en este caso, los potenciales mapeos de los componentes de aplicación (parte derecha de la imagen). Por ejemplo, la primera "DseRule" denota que si el componente "codeComp" se mapea al procesador "micro1" entonces el componente "decoderComp" se no se ha de mapear a ese procesador "micro1".

Unas de las debilidades de esta forma de modelar las variables de diseño era el hecho de que para definir el espacio de exploración de diseño se usasen los mecanismos expresivos anteriormente mostrados ("DseScalarParameter" y "DseAllocationParameter") no estándares. Un análisis más exhaustivo del perfil de MARTE permitió concluir que este perfil sí tenía los mecanismos expresivos necesarios para capturar variables de diseño.

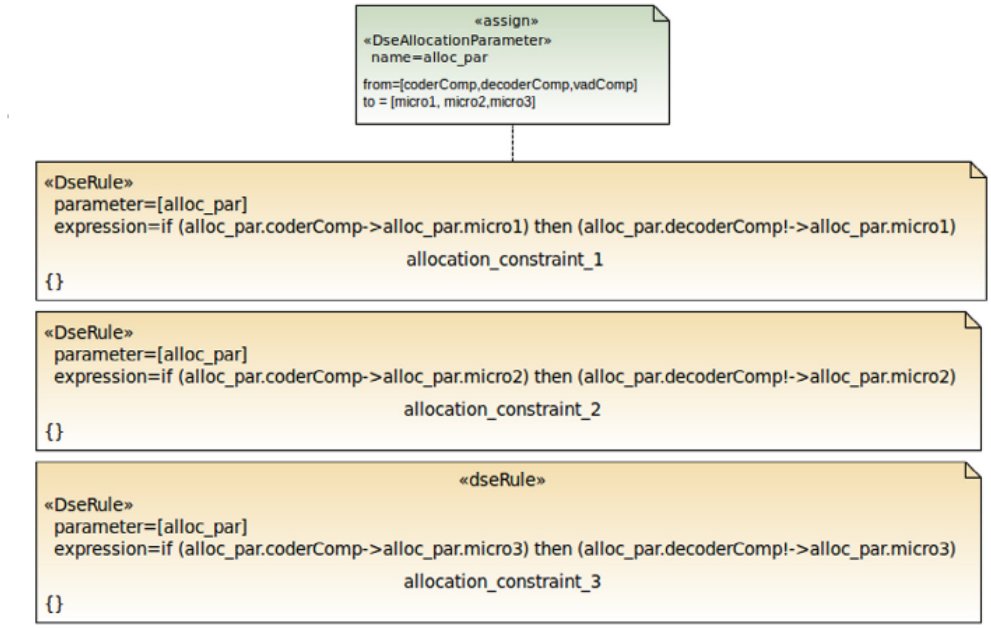


Figura 76 Reglas de diseño

Con esto, la metodología presentada en [b)] para el modelado del espacio de diseño ha evolucionado y se ha sufrido muchas modificaciones (detallada en el manual [qq)]). En [u)] y [w)], la mayoría de esos nuevos conceptos de modelados introducidos en [b)] se eliminaron, usando elementos propios del perfil de MARTE. Estos elementos pertenecen al lenguaje "Value Specification Language", los cuales no fueron considerados con anterioridad en el trabajo [b)].

El lenguaje VSL permite la especificación de parámetros/variables, constantes y expresiones en forma textual. En el trabajo [u)] y [w)], VSL se usa para definir parámetros de exploración a valores de atributos de componentes, anotando todos los potenciales valores que ese parámetro podrá tomar durante el proceso de exploración. Toda esta nueva metodología fue realizada por mí con la colaboración de J. Medina.

La inclusión de estos elementos MARTE provocó los correspondientes cambios en todas las herramientas que fueron implementados por mí, adaptando dichos generadores a los nuevos aspectos de la metodología.

En el ejemplo de la Figura 77 se define una variable de exploración "$busSlots" asociada a al atributo "numberSlots" de componente bus TDMA ("Time Division Multiple Access"). La definición de esta variable (usando el lenguaje VSL) incluye todos los potenciales valores que puede tomar dicha variable durante el proceso de exploración. Para ver cómo evolucionó la notación de estas variables en las diferentes metodologías, nótese la diferencia entre los modelos de esta figura y la de la Figura 75 a).

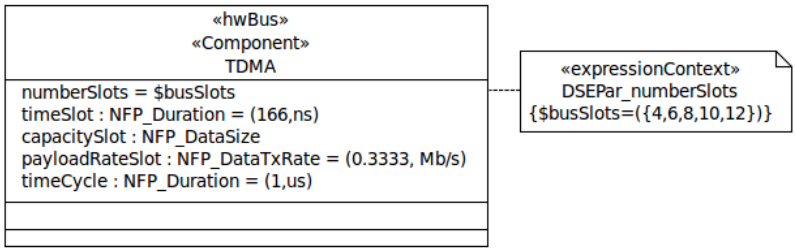


Figura 77 Modelado de un bus TDMA

El modelado de variables de diseño presentado en [b)] únicamente permitía capturar variables de diseño asociada a propiedades de los recursos HW de la plataforma, así como capturar variables que explorasen el mapeo arquitectural de los diferentes componentes de la aplicación en los recursos presentes en la plataforma. Con el fin de eliminar esta restricción expresiva, la metodología UML/MARTE se extendió para poder explorar la estructura concurrente del sistema mediante la exploración de la semántica de comunicación de los canales. De esta manera, el diseñador tendría un abanico más amplio

de aspectos a estudiar y evaluar de sistema, con el fin de tener una visión más amplia de sus potenciales decisiones.

Para más detalles [qq)] y [tt)].

En la sección *Exploración del Espacio de Diseño* del capítulo *Ejemplos de Uso y Resultados* se mostrará, más en detalle, un ejemplo.

Este trabajo se realizó con la colaboración de J. Medina, F. Herrera, H. Posadas y F. Ferrero.

Proyectos en los que este trabajo se desarrolló:

1. *COMPLEX, [32], [xx)]. Director principal: E. Villar; presupuesto: 325588€; periodo: 2009-2013.*
2. *CONTREX, [h)], [ll)], [ww)].*

Publicaciones resultantes:

1. *The COMPLEX methodology for UML/MARTE modeling and design-space exploration of embedded systems, [b)].*
2. *UML/MARTE Modelling for Design Space Exploration of Mixed-Criticality Systems on top of Time-Predictable HW/SW Platforms, [u)].*
3. *A model-based, single-source approach to design-space exploration and synthesis of mixed-criticality systems, [w)].*
4. *The Complex Eclipse Framework for UML/MATE Specification and design Space Exploration of Embedded Systems, [ff)].*
5. *An Embedded System Modelling Methodology for Design Space Exploration, [gg)].*

Capítulo VI
Ejemplos de Uso y Resultados

En este capítulo se van a mostrar ejemplos de uso que van a servir para ilustrar algunos de las diferentes partes del entorno de desarrollo presentado en esta tesis. De esta forma se quiere mostrar la versatilidad de dicho entorno, enseñando como de útil puede ser para encontrar la implementación final del sistema a diseñar. Los diferentes ejemplos que se muestran han sido usados en diferentes proyectos de investigación que se mostraron en la sección *Proyectos de Investigación* del capítulo *Introducción*.

También he de incidir en el hecho que estos casos de uso ha sido una labor colaborativa con diferentes personas involucradas en dichos proyectos. En cada sección de este capítulo se han mencionado explícitamente los nombres de las personas encargadas del desarrollo de dicho ejemplo. En todos ellos, yo he realizado el modelado del sistema y, en algunos de ellos, he participado en la implementación o modificación del código utilizado en los mismos. En todo caso, mi aportación en este sentido ha sido especificada en cada sección.

1. Formalización ForSyDe

En esta sección se va a mostrar un ejemplo para ilustrar lo explicado en sección *Formalización de UML/MARTE: ForSyDe* del capítulo *Modelado de Sistemas Electrónicos.* El ejemplo seleccionado para demostrar las capacidades de la solución propuesta consiste en un elemento concurrente que tiene asociado tres "communication media" para recibir datos y dos para el envío.

La Figura 78 muestra la descripción del comportamiento de dicho elemento concurrente. A partir de ese diagrama de actividad, se obtendrá la anotación formal ForSyDe que describirá las propiedades de dicho comportamiento.

Las señales de entrada son abstraídas como $a_1(z)$, $a_2(z)$ y $a_3(z)$; las señales de salida son abstraídas como $a'_1(z)$ y $a'_2(z)$.

Cada caja coloreada de la figura representa un estado implícito del elemento, los cuales están encapsulado entre dos secciones de "AcceptEventAction" consecutivas. Estos estados están identificados mediante la anotación ω_i (i = 0…7). Después de un estado inicial de inicialización, (ω0), el siguiente estado es $g(\omega_0) = \omega_1$. En este nuevo estado, el elemento concurrente espera a recibir datos desde su entrada 1. En este caso, siempre consume el mismo número de datos, los cuales son denotados por la expresión:

$$\upsilon_1(z) = a \ \forall z \in \mathbb{N}_0$$

En este estado, hay una fase condicional que decide el siguiente estado al que ir, en función de las entradas recibidas. Si $g(a_1, S_1) = S_2$, el elemento requiere j datos desde la entrada 2:

$$v_2(0) = \gamma(\omega_2) = i, i \in \mathbb{N}_0$$

Generando el correspondiente número de datos:

$$v_1'(0) = length\left(function2(a_2(0), \omega_2)\right) = length\left(a_1'(0)\right) = j, j \in \mathbb{N}$$

Después, el elemento concurrente retorna a su estado inicial $g(\omega_2) = \omega_0$.

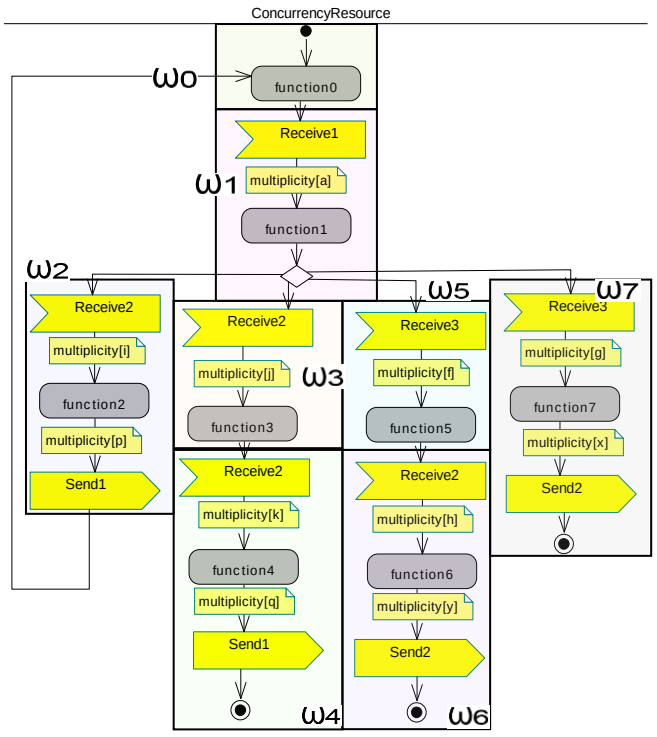


Figura 78 Descripción de comportamiento

En el caso que $g(a_2, \omega_2) = \omega_3$, los datos consumidos son:

$$v_2(0) = \gamma(\omega_3) = j, j \in \mathbb{N}_0$$

Y el nuevo estado es $g(\omega_3) = \omega_4$. En este estado S_4, los datos consumidos son:

$$v_2(1) = \gamma(\omega_4) = k, k \in \mathbb{N}$$

Y los nuevos datos generados son:

$$v_1'(0) = length\left(function4(a_2(1), \omega_4)\right) = length\left(a_1'(0)\right) = q\ , q \in \mathbb{N}$$

Cuando el nuevo estado obtenido es $g(a_1, \omega_1) = \omega_7$, el número de datos consumidos es:

$$v_3(0) = \gamma\ (\omega_7) = g\ , g \in \mathbb{N}$$

Los datos generados son:

$$v_2'(0) = length\left(function6(a_2(0), \omega_6)\right) = length\left(a_2'(0)\right) = y\ , y \in \mathbb{N}$$

Finalmente, si $g(a_1, \omega_1) = \omega_5$, se consumen:

$$v_3(0) = \gamma\ (\omega_5) = f\ , f \in \mathbb{N}$$

Desde este estado, el siguiente estado es $g(\omega_5) = \omega_6$. Los datos consumidos son

$$v_2(0) = \gamma\ (\omega_6) = h\ , h \in \mathbb{N}$$

Y los datos enviados son denotados por la expresión:

$$v_2'(0) = length\left(function6(a_2(0), \omega_6)\right) = length\left(a_2'(0)\right) = y\ , y \in \mathbb{N}$$

2. Simulación Funcional

Como se mostró en *Modelo Atemporal* de la sección *Análisis: Simulación funcional* de capítulo *Entorno de Desarrollo: Integración de Herramientas*, se desarrolló un generador de código SystemC que permitiera la obtención de la correspondiente especificación ejecutable, a partir de las reglas que relacionan MARTE y SystemC mostradas en los trabajos [e)], [i)], [j)], [m)], [n)] y [jj)]. La estructura de esta especificación viene dada por los módulos, los hilos concurrentes y puertos acorde con la estructura capturada en el modelo.

La metodología también permitía la obtención de una estructura de la funcionalidad asociada a cada elemento concurrente del sistema. Esta estructura se definía en bloques funcionales, poniendo énfasis en los segmentos de entrada salida de datos (accesos a canal), según la estructura mostrada en la Figura 21.

Esta metodología se aplicó a un decodificador de video, cuya implementación en SystemC fue proporcionada por F. Herrera. La Figura 79 muestra una parte de dicho sistema, centrándose en el componente MBG ("MacroBlock Generation"), el más complejo del sistema. En dicha figura se puede ver la jerarquía del sistema, los elementos concurrentes ("concurrency resources" como se mostró en la Figura 20) y los canales que se usarán para conectar dichos elementos concurrentes ("communication media" de la Figura 20).

Todos los componentes que se comunican con el MBG se muestran en la Figura 79. Estos son el MPEG "frame decoder" (framedec), el "inverse scanning" (IS) y el componente de reconstrucción DC (DCR). En este caso, todas las funcionalidades son concurrentes. Estos componentes están conectados por medio de los respectivos "communication media".

Cada uno de estos elementos tiene asociado su correspondiente diagrama de actividad para describir su funcionalidad interna. Específicamente, la Figura 80 muestra el diagrama de actividad del componente MGB.

Como se puede ver en las secciones *Formalización de UML/MARTE: ForSyDe* del capítulo *Modelado de Sistemas Electrónicos* y *Modelo Atemporal* anteriormente mencionado, dicho diagrama de actividad se pude abstraerse como un constructor "UMealy" de ForSyDe.

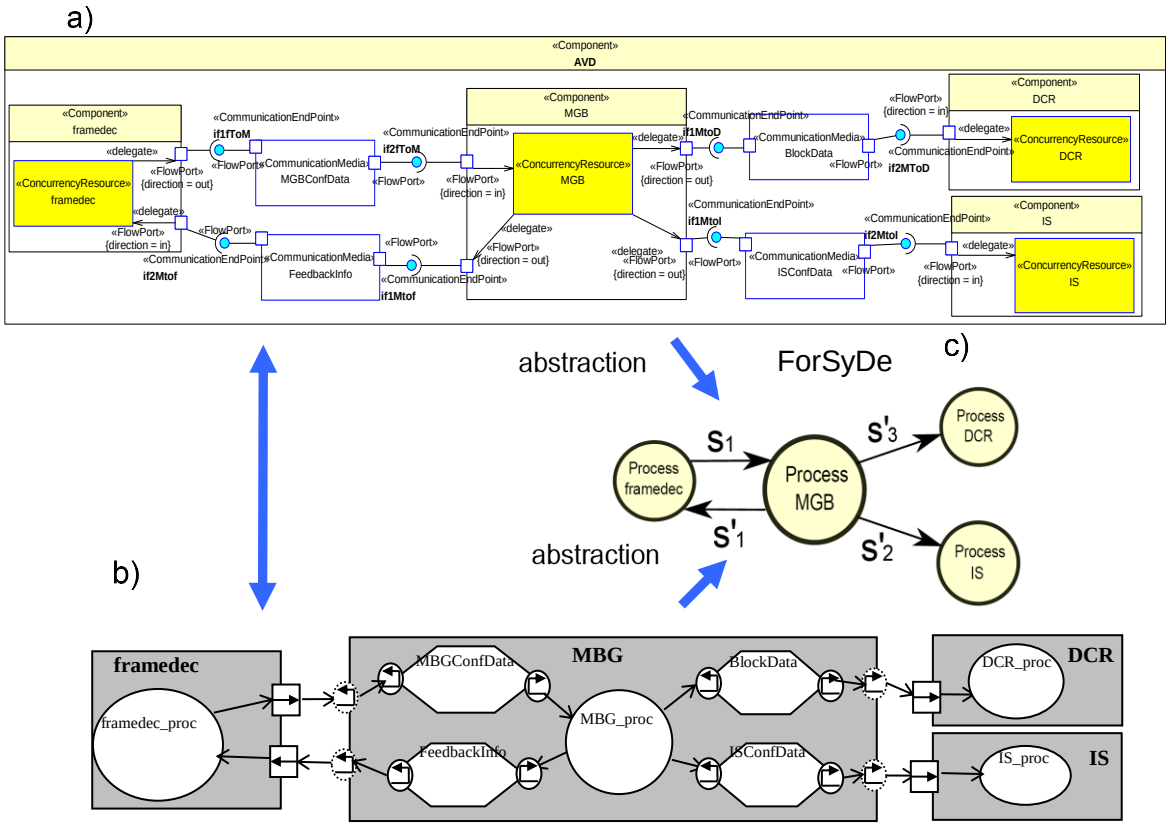


Figura 79 Ejemplo AVD

La abstracción divide las funciones *f, g*, y γ en funciones dependientes de cada estado. Por ejemplo, la función "f" es dividida en 6, ya que de los 8 estados en se pueden identificar en la Figura 80 (mostrados en la Figura 81) 2 no producen salidas.

La función siguiente estado "g" es definida de forma directa en los casos de transiciones no condicionales. En caso contrario, es definida como una función Booleana (como en el caso de $g(\omega_7)$), o como parte de una estado completo ω, como por ejemplo "escape" para el $g(\omega_4)$.

En este ejemplo (Figura 81), las funciones "γ" son constantes para cada estado. Por tanto las funciones de partición son fijas en cada estado. Un ejemplo es el primer estado, donde la partición de la señal ForSyDe de entrada es $\upsilon_{s1}(i)=6$ (independiente de i y solamente dependiente del estado), lo cual denota que un secuencia de 6 (denotados por el valor de la "multiplicity") eventos son leídos del "AcceptEventAction" "MBGConfData.receive". A esto se le corresponde con 6 accesos al correspondiente canal "MBGConfData" (línea 5 del código SystemC).

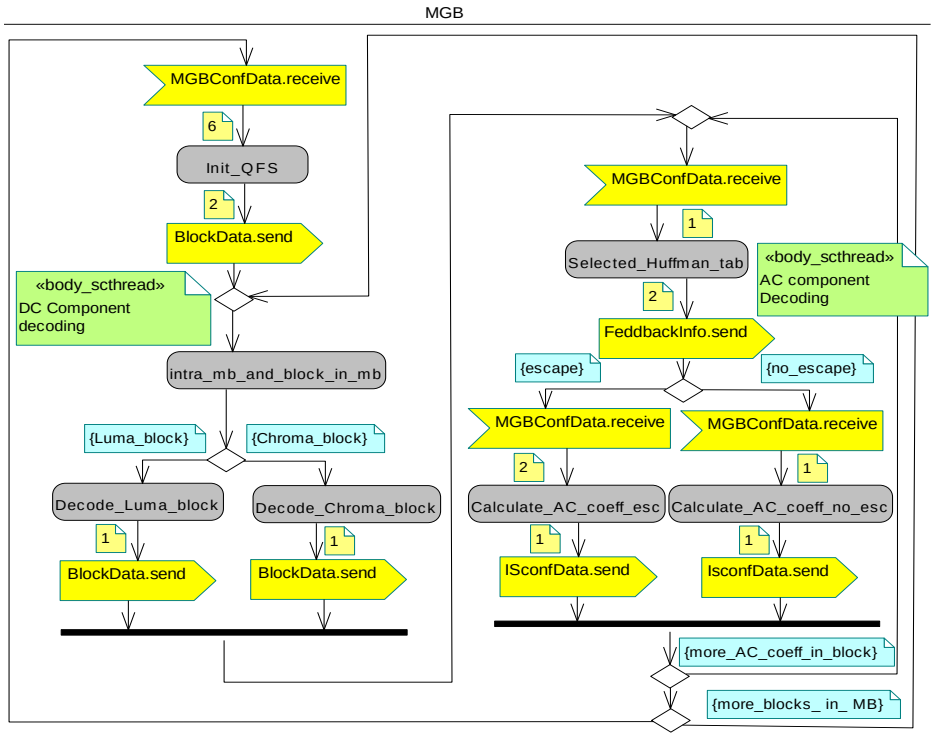


Figura 80 Descripción funcional de MGB

Respecto a las salidas, las señales de salida s'$_1$, s'$_2$ and s'$_3$ abstraen la secuencia de valores enviados por el proceso "MBG", a través de los canales "FeedbackInfo", "BlockData", and "ISConfData".

En cada computación i-th, los procesos consumen/producen sub-señales de las señales ForSyDe de entrada/salida de "MBG". Como ejemplo, las sub-señales s_1 son denotadas por a_{1i}. Las sub-señales de las señales de salida s'$_m$ son denotadas como a_m'i, con m=1, 2, 3 en este caso.

Los estados son modelados como un conjunto discreto y finito de W de estados $\omega_i \in W$ donde i = 0, 1…7. La abstracción es flexible en el sentido que esta partición de estados es flexible. Por ejemplo, una partición del estado 7 podrá haber sido hecha si los estados 5 y 6 hubieran adquirido las condiciones de las transiciones de estado asociadas con el estado 7.

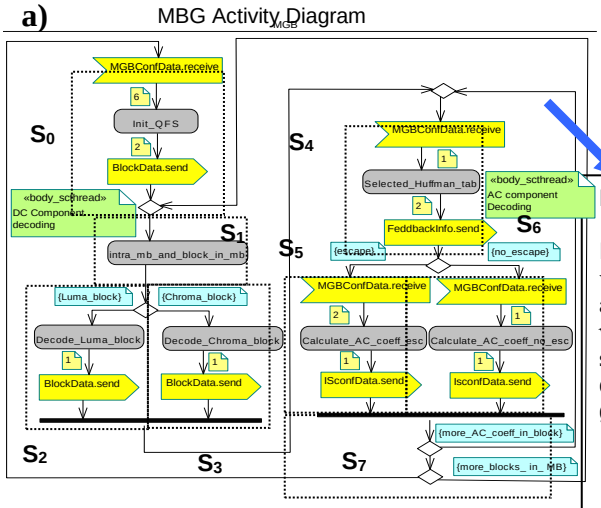
S_i denota el superestado del diagrama de actividad (Figura 21). {V_P, L} denota los valores de todas as variables usadas por el proceso SystemC "MBG_proc" (V_P) y que se localizan en el rango de líneas de código L. Por tanto, hay una asociación entre S_i y

{V_P, L}. Cada ω_i captura esa asociación. Por ejemplo, en ω_4 se captura el superestado S_4 y la ejecución {V_P, (16-18)}.

También se puede ver que los múltiples accesos a canal (para escritura y lectura) son modelados como un bucle para poder acceder las veces que el valor del "mutiplicity" exprese. También del diagrama de actividad se generan estructuras condicionales ("ifs") y bucles condicionados ("whiles").

El hecho que de ambos modelos (UML/MARTE y SystemC) se pueda abstraer en mismo modelo ForSyDe hace que esta transformación sea correcta por construcción.

a)



b)

```
(1)  void MBG::MGB_proc(){
(2)  T1  invar[ ];
(3)  T2  outvar2[ ]; T3 outvar3[ ]; T4 outvar4[ ];
(4)  while (true) {
(5)                  for(int  i=0;i<6;i++)  invar1[i]=
MBGConfData.read();
(6)   Init_QFS(invar1, outvar4);
(7)   for(int i=0;i<2;i++) BlockData.write(outvar4);
(8)  do {
(9)    if (Lumablock()) {
(10)    outvar4[0] = Decode_Luma_block ();
(11)    BlockData.write (outvar4[0]); }
(12)   else {
(13)    outvar4[0] = Decode_Chroma_block();
(14)    BlockData.write (outvar4[0]); } // end if
(15)   do {
(16)    invar[0]= MBGConfData.read()
(17)   Selected_Huffman_tab (invar, outvar2);
(18)                  for(int  i=0;i<2;i++)
FeedbackInfo.write(outvar2[i]);
(19)    if ( escape() ) {
(20)                  for(int  i=0;i<2;i++)  invar[i]  =
MBGConfData.read();
(21)                Calculate_AC_coeff_esc  (invar,
outvar4[0]);
(22)      ISConfData.write(outvar4); }
(23)    else {
(24)      invar[0] = MBGConfData.read();
(25)            Calculate_AC_coeff_no_esc  (invar,
outvar4[0]);
(26)      ISConfData.write(outvar4); } // end if
(27)   } while( more_AC_coeff_in_block () );
(28)  } while( intra_mb_and_block_in_mb( ) );
(29) }  // end MGB process loop code
(30)} // end MGB process code
```

c)

MGB = mealyU(γ, g, f, ω_0)
MGB(s_1) = <s'_1, s'_2, s'_3>
If ($state_i = \omega_0$) then
$v_{s1}(i) = 6$, $\pi(v_{s1}, s_1) = <a1_i>$
$a3'_i = f_0(a1_i) = Init\_QFS(a1_i)$
$v_{s'3}(i) = 2.\ \pi(v_{s'3}, s'_3) = < a3'_i>$
$state_{i+1} = g(\omega_0) = \omega_1$
elseif ($state_i = \omega_1$) then
$g(w_i) = Decode\_Type\_of\_Block(w_i)$

$$state_{i+1} = g(w_7) = \begin{cases} w_2 & Luma\_Block \\ w_3 & Chroma\_Block \end{cases}$$

elseif ($state_i = \omega_2$) then
$a3'_i = f_2() = Decode\_Luma\_block()$
$v_{s'3}(i) = 1,\ \pi(v_{s'3}, s'_3) = <a3'_i>$
$state_{i+1} = g(\omega_2) = \omega_4$
elseif ($state_i = \omega_3$) then
$a3'_i = f_3() = Decode\_Chroma\_block()$
$v_{s'3}(i) = 1, \pi(v_{s'3}, s'_3) = < a3'_i >$
$state_{i+1} = g(\omega_2) = \omega_4$
elseif ($state_i = \omega_4$) then
$v_{s1}(i) = 1, \pi(v_{s1}, s_1) = <a1_i>$
$a1'_i = f_4(a1_i) =$
Selected_Huffman_tab(a1)
$v_{s'1}(i) = 2, \pi(v_{s'1}, s'_1) = <a1'_i>$

$$state_{i+1} = g(w_7) = \begin{cases} w_5 & escape \\ w_6 & \neg escape \end{cases}$$

elseif ($state_i = \omega_5$) then
$v_{s1}(i) = 2,\ \pi(v_{s1}, s_1) = <a1_i>$
$a2'_i = f_5(a1_i) = Calculate\_AC\_coeff\_esc(a1_i)$
$v_{s'2}(i) = 1,\ \pi(v_{s'2}, s'_2) = <a2'_i>$
$state_{i+1} = g(\omega_5) = \omega_7$
elseif ($state_i = \omega_6$) then
$v_{s1}(i) = 1, \pi(v_{s1}, s_1) = <a1_i>$
$a2'_i = f_6(a1_i) =$
Calculate_AC_coeff_no_esc($a1_i$)
$v_{s'2}(i) = 1,\ \pi(v_{s'2}, s'_2) = <a2'_i>$
$state_{i+1} = g(\omega_6) = \omega_7$
elseif (state = ω_7)
$g1(\omega'_i) = more\_AC\_coeff\_in\_block(\omega_i)$
$state_{i+1} = g2(\omega'_i) = more\_blocks\_in\_MB(\omega'_i)$

$$state'_{i+1} = g(w'_i) = \begin{cases} w'_0 & \neg g1 \wedge \neg g2 \\ w'_1 & \neg g1 \wedge g2 \\ w'_4 & g1 \end{cases}$$

Figura 81 Código SystemC y modelo ForSyDe de la funcionalidad de MBG

2.1 UML/MARTE-ForSyDe/SystemC

Para validar todo lo mostrado en *Modelo Atemporal* de la sección *Análisis: Simulación funcional* del capítulo *Entorno de Desarrollo: Integración de Herramientas* de la subsección para la relación MARTE- ForSyDe/SystemC se desarrolló un ejemplo que mostrase la equivalencia semántica entre dos modelos UML/MARTE, en lo que a SDF se refiere. La Figura 82 muestra una parte del ejemplo, el componente "CompositeAvarager" modelado bajo el patrón 1 descrito en subsección *Patrón 1: Flujo de datos Explícito*. La Figura 83 muestra el mismo componente modelado bajo el patrón 2 (*Patrón 2: Llamadas a función*).

Las herramientas descritas en *Relación UML/MARTE-ForSyDe/SystemC* de *Modelo Atemporal* generaron, a partir de ambos modelos de UML/MARTE, el mismo modelo ForSyDe/SystemC (los archivos fuente asociados) y el makefile asociado para su compilación. Alrededor de 200 líneas de código de generaron en 5 segundos sobre un portátil i5 a 1.7 GHz y con 4GB de RAM.

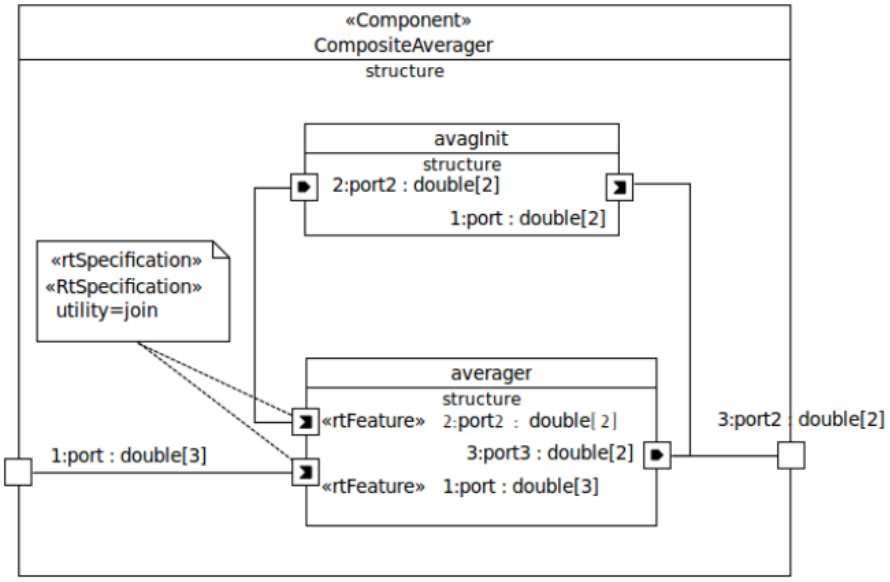


Figura 82 Versión 1 del ejemplo sobre el patrón 1.

Se observó como la estructura del modelo del patrón 1 coincide con la del modelo de ForSyDe/SystemC previsto. Sin embargo, esto no sucede para el segundo caso. El modelo del patrón 2 refleja una perspectiva diferente en la forma de capturar el sistema. El componente "Average" presenta un único puerto que proporciona un interfaz, necesario para disparar su función asociada "average_func". La computación de esta

función solo puede ser hecha cuando recibe los datos necesarios des de “avagInit” y de otro componente no mostrado en la Figura 83.

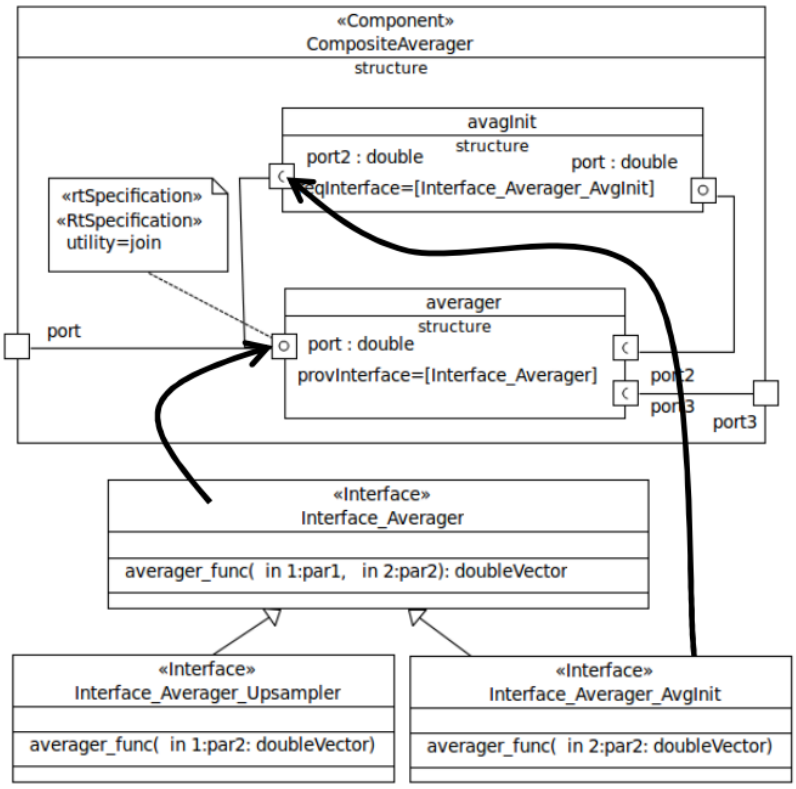


Figura 83 Versión 2 del ejemplo sobre el patrón 2.

El ejemplo que aquí se usa fue proporcionado por S. Hosein y K. Rosvall dentro del proyecto CONTREX ([h)], [ww)]).

3. Canales y Concurrencia

Como se explicó en la sección *Canales y Concurrencia* del capítulo *Modelado de Sistemas Electrónicos,* dependiendo de las propiedades de los canales, interfaces y del propio componente de aplicación se puede derivar una cierta estructura concurrente. Esto se muestra en el ejemplo de un codificador H264; con este ejemplo, se obtienen estructuras concurrentes muy diferentes, lo que permite estudiar el impacto de ellas sobre el rendimiento del sistema. El código de este ejemplo fue proporcionado por la empresa IMEC y adaptado por A. Nicolás y H. Posadas, en el contexto del proyecto PHARAON ([g)], [aaa)]). La realización de este ejemplo fue una tarea compartida con A. Nicolás, H. Posadas.

Para entender los siguientes casos de uso que se van a presentar hay que explicar la Figura 84. Ahí se muestran el código de flechas que representan una semántica de concurrencia específica. Esto será útil para los ejemplos mostrados en la Figura 86 y la Figura 88.

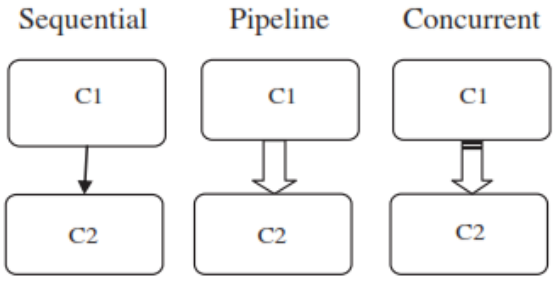


Figura 84 Tipos de semántica de comunicación

La Figura 85 muestra el modelo de UML/MARTE de la aplicación de codificador MPEG-4. Esta aplicación se ejecutaré sobre la plataforma OMAP4, la cual posee dos núcleos. El codificador MPEG-4 es un estándar industrial, que consistente en una fase de estimación de movimiento y compensación seguida de fases de transformación y codificación de entropía. El modelo UML creado contiene un conjunto de bloques funcionales: MEMC ("MotionEstimation-MotionCompensation"), TCTU ("TextureCoding-TextureUpdate"), "EntropyCoding" (EC) y "BitstreamPacketizing" (BP).

La implementación del codificador H264 utilizado en este caso de uso permite establecer configuraciones de sistema diferentes mediante la modificando de la semántica de los canales del modelo (Figura 85). A partir de una implementación secuencial inicial,

la semántica de los canales permite modificar esta estructura iniciales en otras en las que diferentes componentes pueden ejecutarse en paralelo.

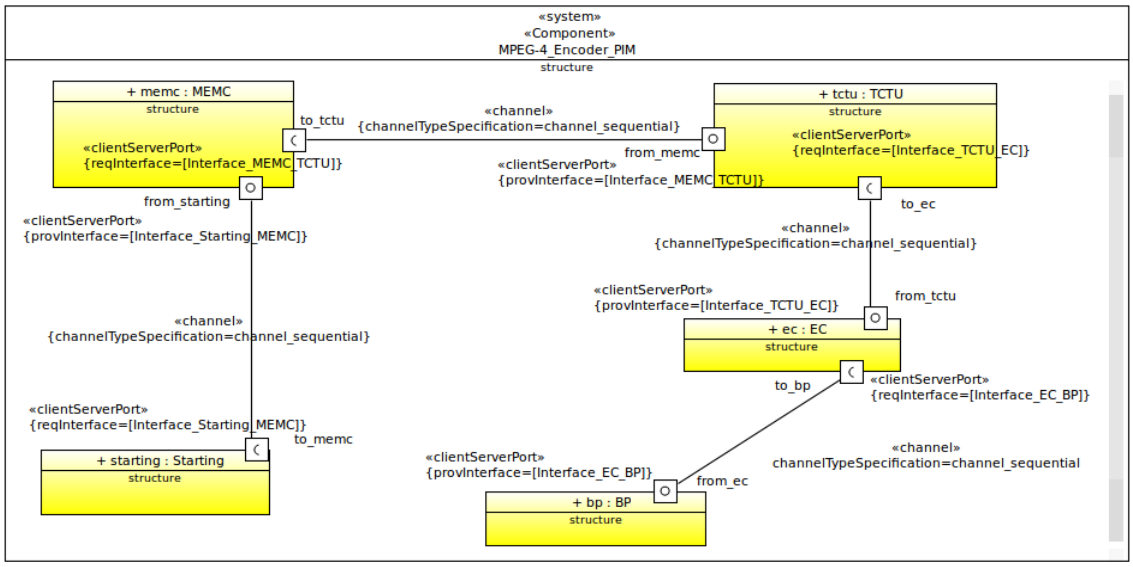
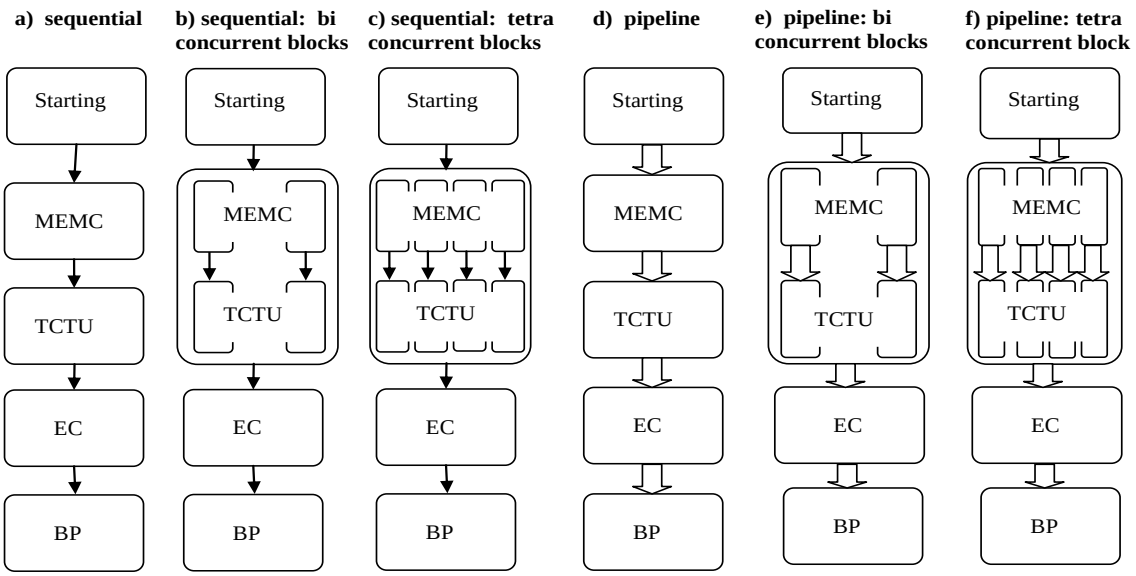


Figura 85 Modelo de la aplicación del H264.

Una vez que se ha hecho el modelo de la Figura 85, se evalúan las diferentes implementaciones cambiando la semántica de los canales y las interfaces utilizadas por cada componente. Como resultado, se definió un espacio de exploración de diseño que abarca múltiples arquitecturas concurrentes. Estas se muestran en la Figura 86.



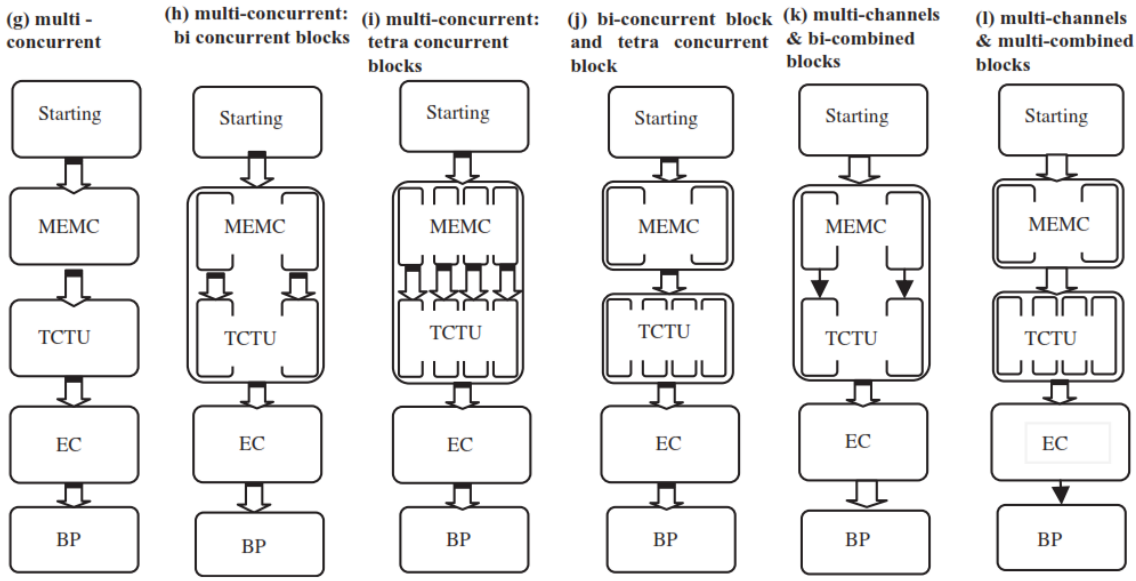


Figura 86 Conjunto de estructuras concurrentes

La Tabla 1 muestra los tiempos de ejecución de las diferentes estructuras mostradas en la Figura 86 sobre la plataforma OMAP4430 (con dos procesadores). Como se puede ver, cada una de las anteriores estructuras concurrentes tiene impacto en el rendimiento del sistema.

Tabla 1 Resultados de la ejecución de los casos mostrado en la Figura 86.

Caso	Real (s)	Mejora
a) sequential	21.1	Referencia
b) sequential: bi concurrent blocks	13.0	1.6x
c) sequential: tetra concurrent blocks	13.2	1.6x
d) pipeline	15.7	1.3x
e) pipeline: bi concurrent blocks	11.4	1.8x
f) pipeline: tetra concurrent block	11.1	1.9x
g) multi concurrent	11.2	1.9x
h) multi concurrent: bi concurrent blocks	11.0	1.9x
i) multi concurrent: tetra concurrent blocks	11.1	1.9x
j) bi concurrent block and tetra concurrent block	12.9	1.6x
k) bi combined blocks	11.9	1.6x
l) multi combined blocks	11.4	1.6x

Ejemplos como estos muestran que no es trivial, en lo que al impacto en el rendimiento se refiere, la selección de las propiedades de los canales y de la estructura concurrente que se puede derivar de estas. Esto prueba la utilidad de poder explorar este tipo de propiedades de una manera rápida y automática.

Otro escenario de aplicación es ver el impacto que tiene en el rendimiento del sistema la herencia de interfaces mostrada en la sección *Canales y Concurrencia* del capítulo *Modelado de Sistemas Electrónicos*, como forma de unir flujos concurrentes independientes. Para mostrar la aplicación de esta parte de la metodología, se usó, como ejemplo de uso, un sistema de estéreo visión; dos imágenes se rectifican inicialmente para permitir una posterior comparación para poder extraer la posición de diferentes objetos en la imagen, respecto al observador. El código de este ejemplo fue proporcionado por la empresa TeDeSys, siendo modificado por A. Nicolás, principalmente, y por mí.

Ambas imágenes son procesadas en paralelo. Para ello, se definen dos flujos paralelos para realizar este procesado, una para la imagen derecha y otra para la imagen izquierda. Primero se realiza un pre-procesado para ver si la imagen tiene calidad suficiente; detecta si la imagen tiene más del 80% de los pixeles en negro. Si ese número se da, se aborta la aplicación.

Después de esta fase, comienza el algoritmo propio de la visión estéreo. Se computa la densidad de disparidad o mapa de profundidad de ambas imágenes. La estructura y modelo de esta aplicación se muestra en la Figura 87, que muestra siete componentes.

La funcionalidad implementada en este test consiste en procesar cuatro pares de imágenes. Estas imágenes son procesadas de acuerdo a diferentes alternativas de diseño en función de diferentes tipos de configuraciones de comunicación (Figura 88). Estas configuraciones vienen definidas por las propiedades de los canales y de las interfaces asociadas. Todos los canales están involucrados en la especificación de la estructura concurrente. Sin embargo, alguno de ellos tiene que ser identificados explícitamente para caracterizar las cinco alternativas de diseño consideradas en este caso. Estos canales son el “stereoSegmentationRight-stereoMatching” (sts-stm_right), “stereoSegmentationLeft-stereoMatching” (sts-stm_left), “starting-stereoSegmentationRight” (sta_ste_right) y “starting-stereoSegmentationLeft” (sta_ste_left).

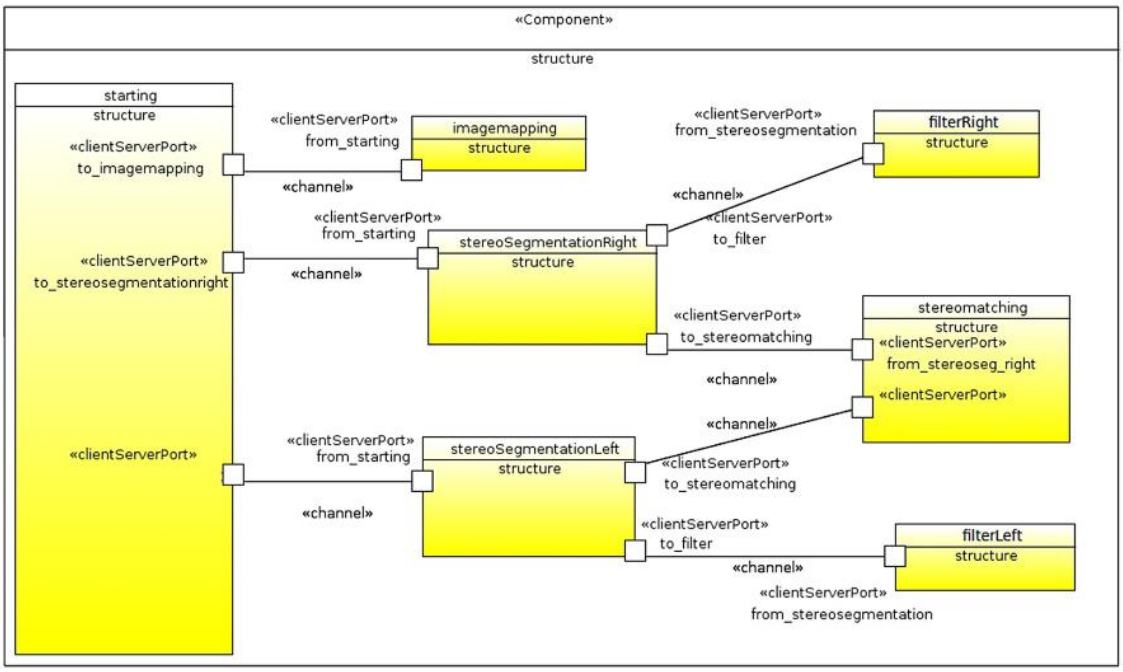


Figura 87 Ejemplo de estéreo visión, versión 1.

En un primer escenario, los pares de imágenes son procesados de manera secuencial, primero la imagen derecha y luego la izquierda. Para hacer esto, todos los canales son puramente secuenciales (Figura 88, caso 1). Una vez que el componente "stereoMatching" tiene ambas imágenes disponibles mediante la implementación de unión de servicios descrita en la sección *Canales y Concurrencia* del capítulo *Modelado de Sistemas Electrónicos*. Esto se denota en la Figura 88 mediante las líneas de segmentos para los canales "sts-stm_right" y "sts-stm_left". En este punto, la comparación de las imágenes se realiza, comenzando el procesamiento de un nuevo par de imágenes.

En el segundo caso, cada imagen del par es procesada en paralelo con la otra (Figura 88, caso 2). El canal "sta_ste_right" y el canal "sta_ste_left" se definen como fue descrito en *Interfaces* de la sección anteriormente mencionada. De esta forma, se puede procesar el siguiente par de imágenes sin tener que parar. En este caso, aunque presenta cierto paralelismo, el diseño aún presenta estructura secuencial: los pares de imágenes son procesados de forma secuencial.

En los siguientes casos, la estructura concurrente implementada es de pipeline (Figura 88, casos 3, 4 y 5).

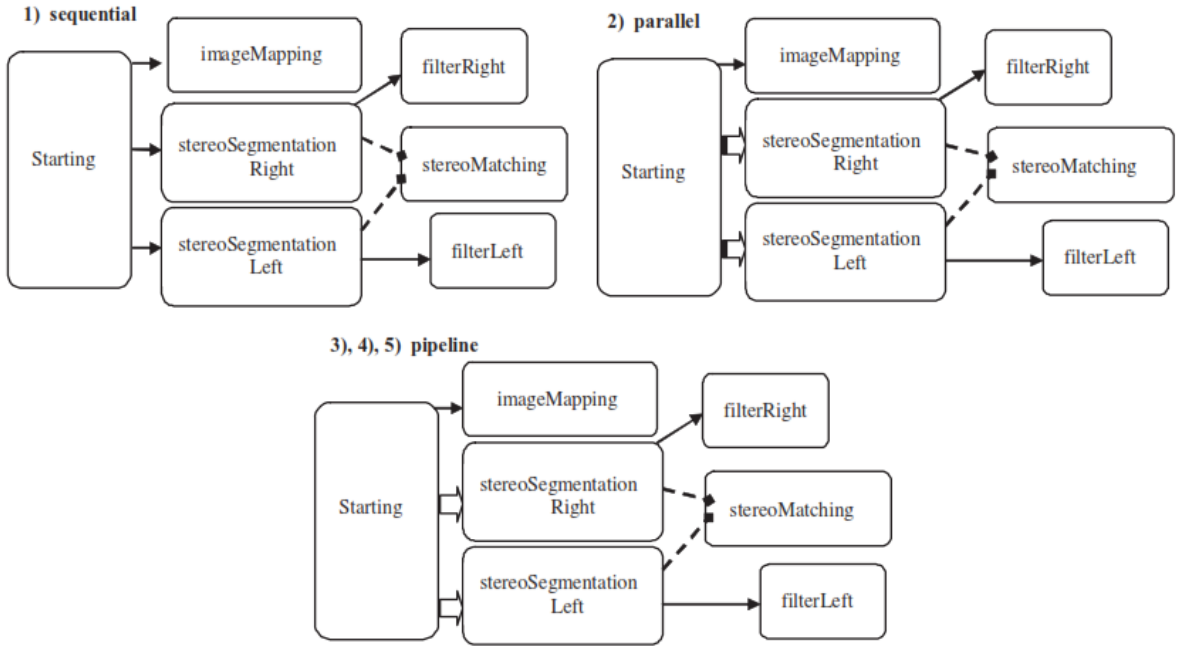


Figura 88 Diferentes estructuras concurrentes a explorar

En estos casos se han considerado diferentes escenarios en lo que propiedades de los componentes. En el caso 3, los componentes solo pueden procesar una imagen. En el segundo caso, los componentes pueden computar en paralelo 2 y 4 imágenes respectivamente. Esto se hace modificando un valor de una de sus propiedades (sección *Canales y Concurrencia*). De esta forma, múltiples imágenes son procesadas en el mismo componente durante todo proceso global de cómputo.

Tabla 2 Resultados de la ejecución de los casos de la Figura 88.

Caso	Real (s)	Mejora
1) sequential	78.6	Referencia
2) parallel	45.1	1.7x
3) pipeline_1	41.5	1.9x
4) pipeline_2	42.1	1.9x
5) pipeline_4	45.4	1.7x

Los resultados obtenidos se muestran en la Tabla 2, después de realizar el proceso de síntesis de SW y de ser ejecutados en una Panda (que tiene dos núcleos).

Los primeros tres resultados pueden ser fácilmente entendidos de acuerdo a la semántica de los canales considerados. Sin embargo, los casos 4 y 5 son más complicados. Primero, los cuatro flujos concurrentes del caso 5 tienen ciertas limitaciones en lo que a

paralelismo se refiere, ya que la Panda solo tiene dos núcleos. Además, puede verse que los casos 4 y 5 son peores que el caso 3, debido a que se generan más hilos de ejecución concurrentes. Esto es causado por el ratio mayor de accesos a memoria, necesarios para obtener los datos requeridos cuando estos no están disponibles en las caches de datos.

4. Síntesis de HW: OpenMAX

Como prueba de concepto que mostrará lo beneficios de la infraestructura mostrada en *OpenMAX* de la sección *Generación de código: Síntesis de HW* del capítulo *Entorno de Desarrollo: Integración de Herramientas* se usó el sistema que se muestra en la Figura 89 (realizada por D. de la Fuente).

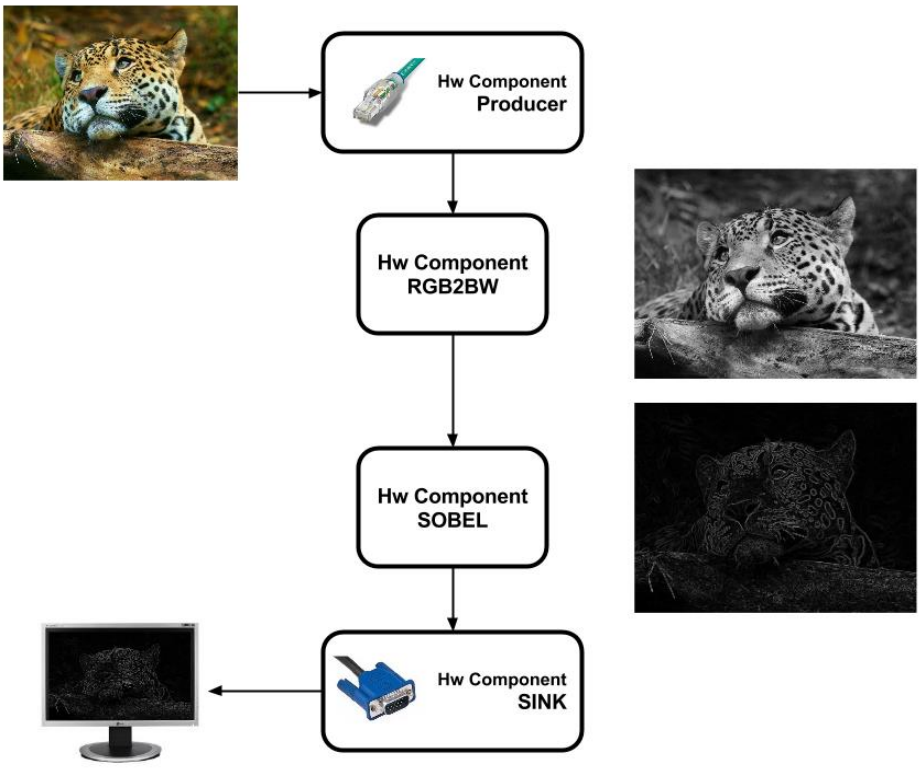


Figura 89 Aplicación SOBEL

El objetivo de este caso de uso es aplicar el algoritmo SOBEL para detectar los bordes de una secuencia de imágenes. Para implementar este algoritmo, se requiere de imágenes en escala de grises. Para esto, se aplica inicialmente un filtro RGB blanco y negro. Finalmente, la imagen será mostrada en un monitor. Este caso de uso fue proporcionado y desarrollado por las personas del grupo de departamento ARCO de la Universidad de Castilla-La Mancha, en el contexto del proyecto DREAMS ([zz)]).

Todas las tareas computacionales se realizan sobre una FPGA, en concreto una Xilinx modelo Virtex-5 XC5VFX70T.

La aplicación (Figura 89) consiste en cuatro componentes OpenMAX:

- El primero recibe.
- El segundo convierte la imagen a escala de grises.
- El algoritmo SOBEL se placa en el tercer paso.
- En el último se muestra la imagen resultado en un monitor.

La Figura 90 (realizada por D. de la Fuente basándose en una imagen mía) muestra el modelo UML/MARTE de la aplicación SOBEL/OpenMAX. Todos los componentes tienen asociados una estimación del tiempo de ejecución, necesaria para la fase de simulación que denotado como muestra la Figura 33.

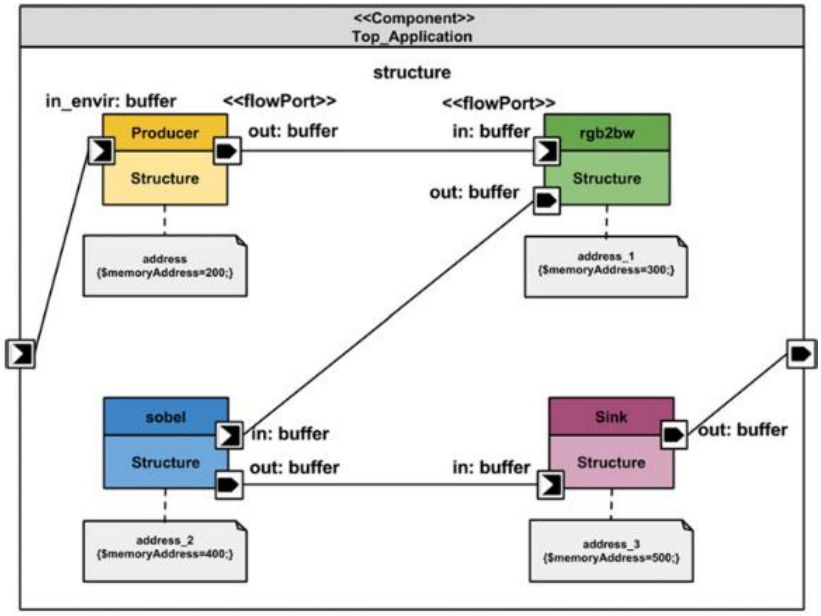


Figura 90 Modelo UML/MARTE del sistema SOBEL.

Una vez hecho esto, se especifica la naturaleza HW/SW del componente OpenMAX, como se muestra en la Figura 91 (realizada por D. de la Fuente basándose en una imagen mía).

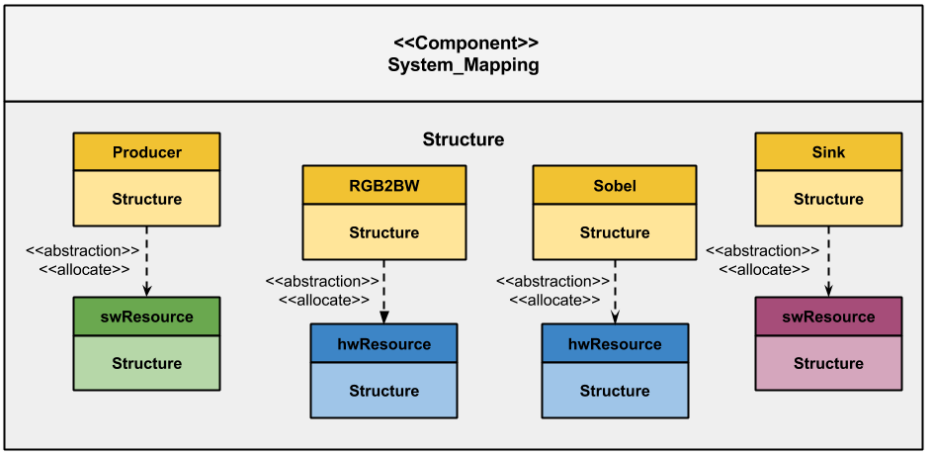


Figura 91 Mapeo HW/SW de los componentes OpenMAX

Una vez que se han especificado todas estas características, se puede realizar la fase de simulación SystemC, generando dicho código directamente desde el modelo UML/MARTE. Aquí, el modelo se modifica manualmente, obteniendo una nueva configuración del sistema que será simulada. Esto representa un proceso semiautomático de exploración del espacio de diseño del sistema.

De este análisis se puede obtener una serie de conclusiones:

- Usando ráfagas de 64 palabras en lugar de 16, el tiempo de ejecución del sistema mejora un 67%.
- Fijado el tamaño de ráfaga a 64 palabras, la mejor distribución de búferes fue con cuatro búferes de 4KB para cada componente (dos para las entradas y dos para las salidas), configurando una estructura de ping-pong. Con esto se mejora el rendimiento en otro 14.37%.
- Si los componentes transfieren el contenido de los búferes en paquetes tan pronto como ellos están disponibles en sus memorias de salida (modo “continuos”), el tiempo de ejecución se reduce en otro 4.18%.
- Se estima que el rendimiento es de 68.92 fps (“frame per second”).

Finalmente, con la configuración final del sistema, se establece la generación del correspondiente VHDL asociado a los componentes OpenMAX definidos en el modelo, según en esquema mostrado en Figura 92 (realizada por D. de la Fuente).

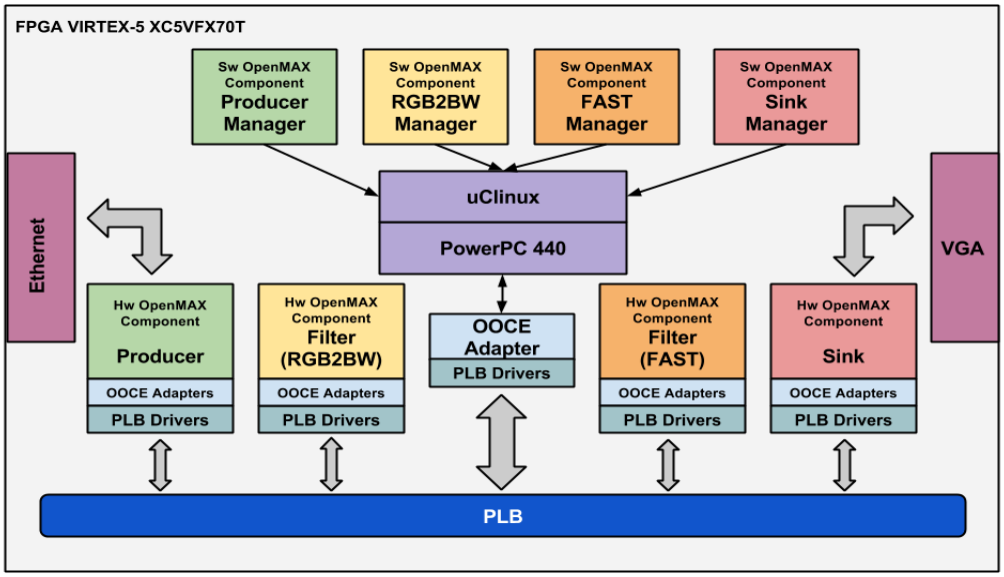


Figura 92 Contenido de la FPGA después de la síntesis.

Como pude verse en la Figura 92, se ha generado un componente SW, que actúa como manejador, por cada componente HW. Para las comunicaciones entre los componentes se utiliza como canal compartido un bus PLB de 64 bits. El generador crea los adaptadores PLB necesarios para cada componente, incluyendo lo elementos necesario para establecer las comunicación HW/HW y HW/SW, a través del middleware OOCE [d)].

Las líneas de código generadas se muestran en la Tabla 3.

Tabla 3 Líneas de código generadas en el ejemplo de OpenMAX.

Elemento Generado	Producer	RGB2BW	SOBEL	Sink
Driver del Bus	381	381	381	381
Adaptadores OOCE	763	763	763	763
FIFOs	278	278	278	278
Adaptador OpenMAX	815	777	777	672
Top	271	247	247	239
SW componente manejador	3004	3004	3004	3004
Total	5512	5450	5450	5337

El sistema generado fue testeado con una secuencia de imágenes con un tamaño de 640x480 pixeles y que ocupan aproximadamente 300 KB cada una. Cuando el test finalizó, el número de imágenes procesadas por segundo fue de 63.64, lo que implica un error con respecto a la simulación de un 8.29 %.

Para más información del ejemplo, ver [d)].

5. Síntesis de HW: Reconfigurabilidad

Como caso de uso para mostrar toda la infraestructura descrita en la sección *Reconfigurabilidad* del capítulo *Entorno de Desarrollo: Integración de Herramientas* se volvió a usar el ejemplo de la sección anterior: una aplicación basada en el algoritmo SOBEL. La estructura de este nuevo caso de uso se muestra en la Figura 93 (realizada por D. de la Fuente).

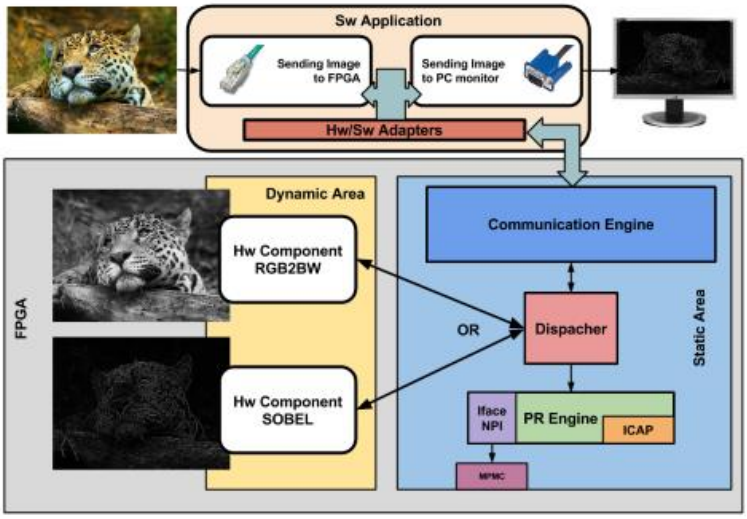


Figura 93 Estructura de la aplicación.

De nuevo, la aplicación consta de cuatro componentes, como en el caso de la sección anterior, realizando la misma funcionalidad asociada.

En este caso, se considera un escenario de funcionamiento en el cual los recursos disponibles para mapear el componente RGB2BW y el filtro SOBEL no son suficientes. Esto implica la necesidad de hacer uso de áreas dinámicas. Gracias a esta consideración, se puede proporcionar un dispositivo que es capaz de reconfigurarse por sí mismo, permitiendo la implementación de diferentes funcionalidades en él. Este caso de uso fue proporcionado y desarrollado por las personas del grupo de departamento ARCO de la Universidad de Castilla-La Mancha, dentro del contexto del proyecto ([zz)]).

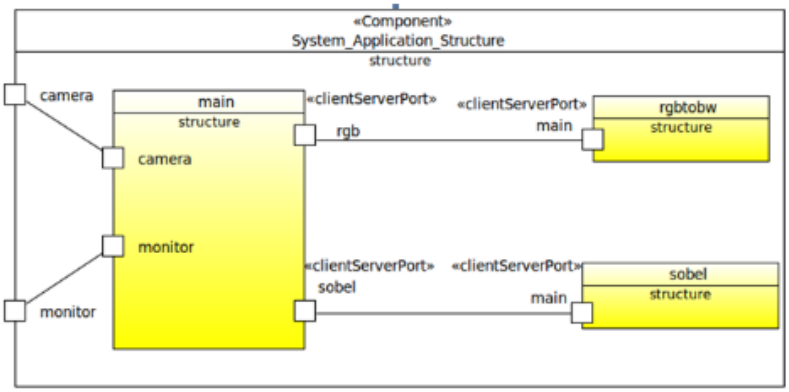


Figura 94 Modelo de la aplicación

En este caso el modelo UML/MARTE de la aplicación se compone de tres componentes, actuando el componente "main" como frontera para la adquisición y envío de las imágenes al entorno del sistema.

Estos componentes se mapean espacios de memoria (Figura 95), para poder realizar el posterior mapeo en los recursos HW de la plataforma (Figura 96). En este caso se puede ver como los componentes RGB2BW y el filtro SOBEL son mapeados a la FPGA.

El componente FPGA es caracterizado como se muestra en la Figura 71.

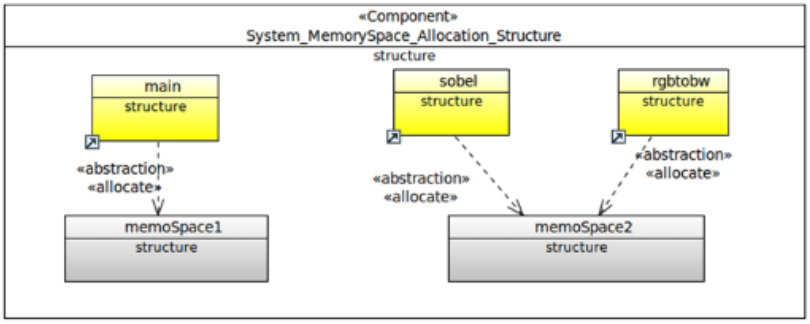


Figura 95 Asociación a espacios de memoria

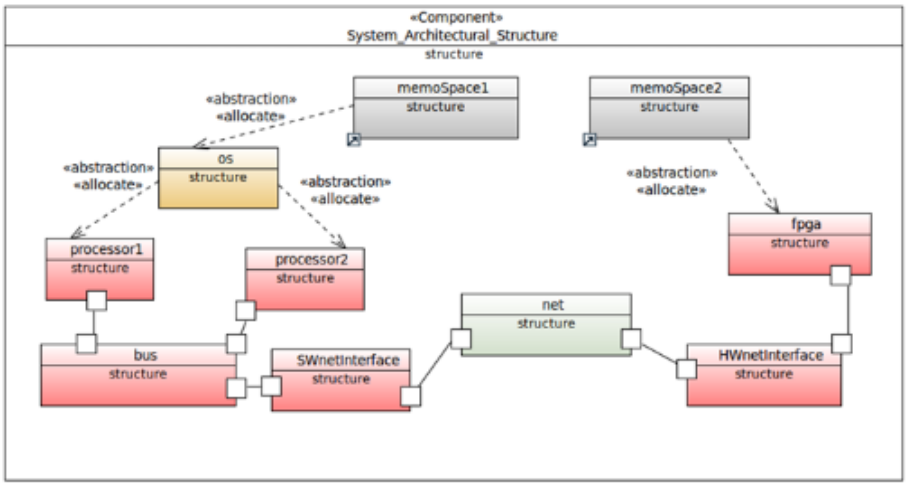


Figura 96 Mapeo arquitectural

A partir de toda la infraestructura desarrollada y explicada en *Reconfigurabilidad* de la sección *Generación de código: Síntesis de HW* del capítulo *Entorno de Desarrollo: Integración de Herramientas*, se realiza una simulación SystemC permitiendo una evaluación temporal y del comportamiento del sistema. En este caso, todos los componentes tienen asociados una estimación del tiempo, la cual es requerida en esta fase. El análisis da un conjunto de resultados que permiten desarrollar un conjunto de conclusiones:

- Si solo se usa el filtro RGB2BW, el rendimiento estimado es de 63 imágenes por segundo.
- Usando la reconfiguración dinámica y el filtro SOBEL, el rendimiento se reduce a 35.27 %.
- La tasa de reconfiguración es de 180.3 MB/s.

Sirva como referencia las líneas de código generadas de forma automática y que se muestran en la Tabla 4.

Tabla 4 Líneas de código generadas en el ejemplo de reconfigurabilidad.

Elemento VHDL	Líneas de Código	Elemento (SystemC)	Líneas de Código
Área estática	4568	Plataforma SystemC	950
Wrapper RGB2BW	1483	RGB2BW	69
Wrapper SOBEL	1771	SOBEL	69
Total	7822	Total	1080

6. Exploración de Recursos HW

En esta sección se va a mostrar el impacto en el rendimiento del sistema al mapear cierta funcionalidad en uno u otro recurso HW de la plataforma.

El sistema de la Figura 97 corresponde a un sistema de visión estereoscópica, pero en una versión diferente de la mostrada en la sección *Canales y Concurrencia* de este capítulo. El código de este ejemplo fue proporcionado por la empresa TeDeSys y modificado por A. Nicolás, principalmente, y por mí, en el contexto del proyecto PHARAON ([g)], [aaa)]).

En este sistema, las imágenes recibidas de dos cámaras se rectifican para poder compararlas con el fin de extraer la posición de diferentes objetos de la imagen con respecto al observador.

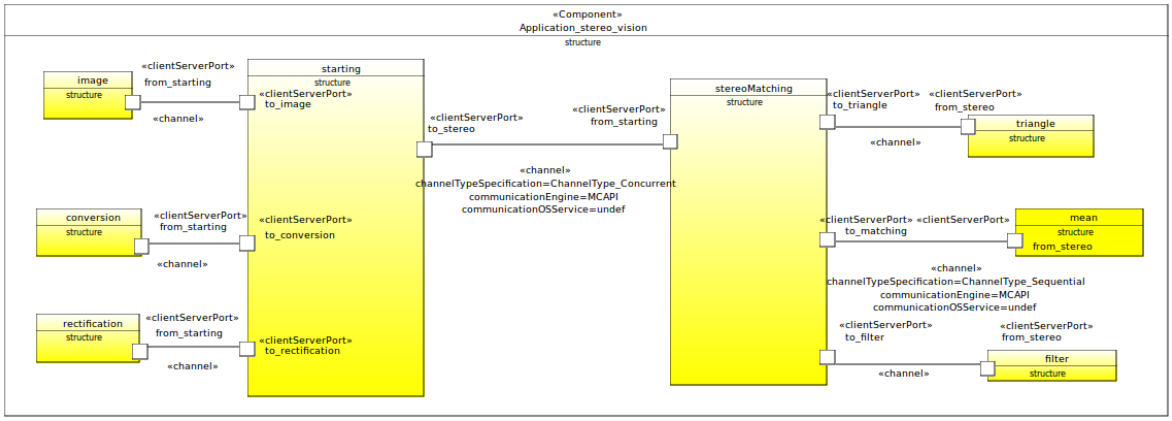


Figura 97 Sistema de visión estereoscópica, versión 2

Los componentes funcionales se asignan a los espacios de memoria (Figura 98) agrupando funcionalidad, los cuales serán asignados a los diferentes recursos HW/SW presentes en la descripción de la plataforma. Por lo tanto, los componentes asociados a un mismo espacio de memoria siempre se manejan juntos.

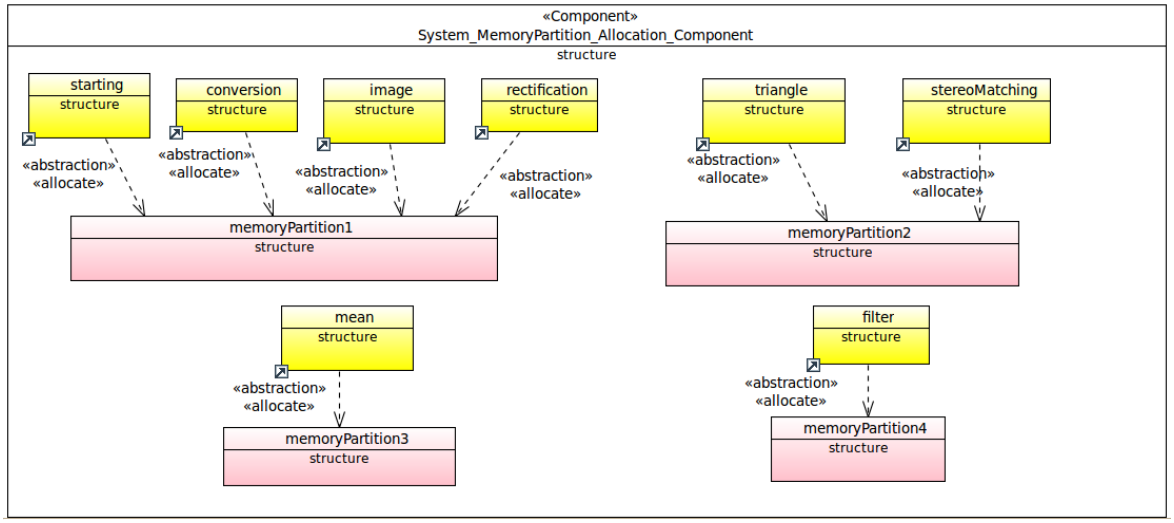


Figura 98 Mapeo de componentes de aplicación a espacios de memoria

Una vez hecho esto, se puede realizar la asignación de esos espacios de memoria a la plataforma. La Figura 99 representa las mismas plataformas mostradas en la Figura 27, que representa una Beagle (que tiene un núcleo y un DSP) y i.MX6 (que tiene cuatro núcleos). En ellas se puede ver los mapeos arquitecturales de los espacios de memoria anteriormente definidos a los diferentes recursos de la plataforma.

En el caso de Figura 99A) se puede ver como los espacios de memoria son mapeados a un procesador y a un DSP. En el segundo caso, los espacios se mapean a los procesadores presentes a través del sistema operativo que los maneja.

La placa Beagle y el mapeo arquitectural mostrado en la (Figura 99A) se han utilizado para realizar experimentos en los que intervenga el DSP y el NEON, en varios escenarios. El NEON es un co-procesador que tienen lo núcleo Cortex. En el caso (A), la aplicación se mapeó en su totalidad en el procesador ARM. En una segunda configuración (B), el componente de "mean" (en el modelo, se mapea su espacio de memoria correspondiente, el llamado “memoryPartition3” de la Figura 98) se asocia al coprocesador NEON del procesador. Con esta asociación, el conjunto de herramientas desarrolladas detecta que un espacio de memoria está mapeado a un procesador que es del tipo Cortex-A. De esta manera, dichas herramientas generan el código necesario para que ese componente pueda ser compilado y ejecutado en dicho co-procesador NEON. La elección del componente “mean” como componente a explorar su mapeo en diferentes recursos HW viene dada porque es el componente que requiere más potencia de cómputo durante su ejecución. En un tercer caso (C), el componente "mean" se asigna al DSP, moviendo el enlace de mapeo del correspondiente al recurso DSP tal como se ve en la

Figura 99A. De la misma forma que en caso anterior, el conjunto de herramientas desarrolladas genera toda la infraestructura de código para compilar y ejecutar dicho componente en el DSP de forma automática.

A)

B)

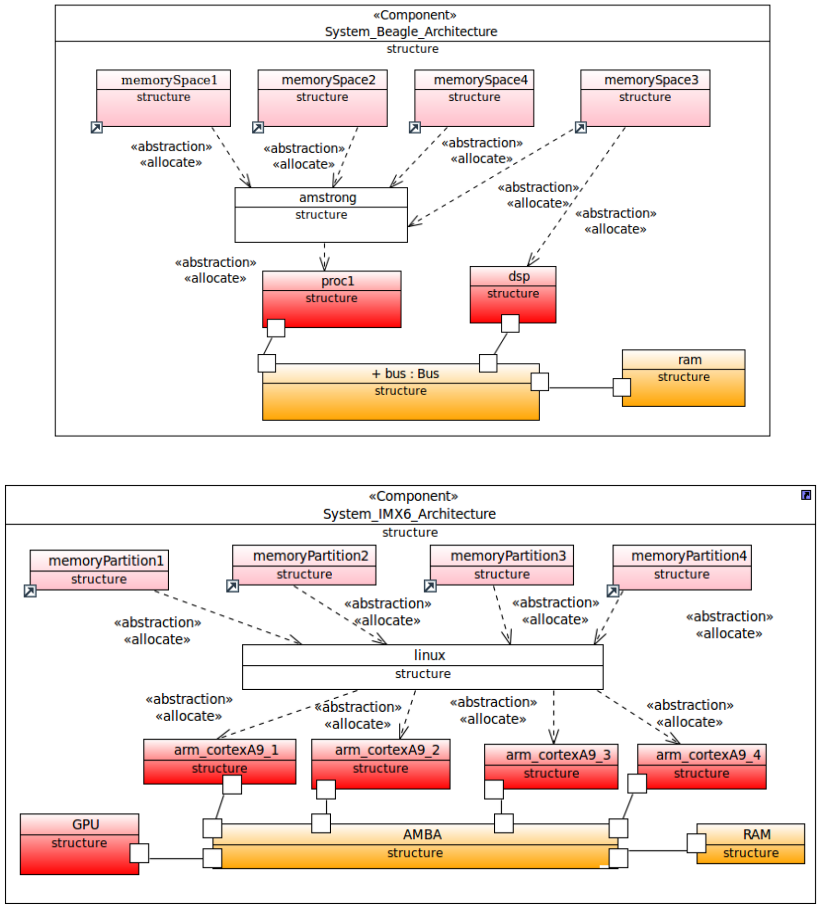


Figura 99 Mapeo arquitectural a diferentes plataformas

En todos los casos anteriores, el flujo de ejecución es secuencial. Sin embargo, en las dos últimas configuraciones (D, E) se añade el paralelismo al sistema. En estos casos, las imágenes se procesan de forma simultáneamente. En la Figura 97, el canal entre los componentes "stereoMatching" y "starting" se modifica; sus propiedades bloqueantes y de almacenamiento son cambiadas de tal forma que se permitan múltiples llamadas concurrentes al servicio de interfaz. Además de este procesado concurrente de la imagen, los espacios de memoria son asignados a los recursos de la plataforma HW de forma dinámica. La Figura 99 A) muestra como el espacio de memoria “memorySpace3” está mapeo al procesador y al DSP a la vez. Esto indica que se ha de generar la infraestructura de código necesaria para este procesado dinámico; las herramientas generan toda la

infraestructura de código que permite la ejecución de componente en el DSP y en el procesador (y en el co-procesador NEON, en este caso).

De esta forma, el componente "mean" se ejecuta de forma dinámica en el recurso HW dependiendo de qué recurso esté libre.

Tabla 5 Resultados de diferentes mapeos sobre la Beagle

Configuración	Original (s)	Mejora
A. ARM	50.58	Referencia
B. ARM+NEON “mean”	25.30	1.99
C. DSP “mean”	45.24	1,11
D. ARM+DSP “mean” dynamically	47.88	1.06
E. ARM+NEON+DSP dynamically	37.09	1,36

La Tabla 5 muestra los resultados obtenidos en los anteriores escenarios. Como valor de referencia para generar dicha tabla se ha tomado en caso A). Por un lado, se observa una mejora con el uso del DSP (caso C). Aunque donde verdaderamente se mejora muy sustancialmente el rendimiento es el caso B, donde la ARM aprovecha al máximo las prestaciones del NEON. A priori podría esperarse que el procesado dinámico ofreciera unas grandes prestaciones. Pero los resultados contradicen está hipótesis. En el caso D, que permite la computación en paralelo en ARM-DSP no proporciona casi ganancia, ya que el procesador ARM es el cuello de botella del cómputo. Por último, en el caso E, lo mismo sucede como en el caso D, pero ahora con el conjunto ARM-NEON es más rápido que DSP, transformándose ahora el DSP en el cuello de botella.

Los resultados obtenidos muestran la diferencia de rendimiento dependiendo de la forma en la que los diferentes dispositivos HW son usados. Sin embargo, no todos los resultados son como el diseñador podría esperar inicialmente, lo que demuestra que el tener un entorno de desarrollo que permite la modificación e implementación de manera fácil y rápida de un sistema es importante, ya que hace que el usuario pueda evaluar todas las alternativas de su diseño sin esfuerzo.

Para la mostrar cómo el entorno de diseño permite la utilización de la GP-GPU se ha seleccionado una aplicación distinta, más adecuada para las características de dicho componente HW. Dicha aplicación se denomina “Yet Another Waveform” (YAW). YAW es un tipo de paquete “Internet Protocol” (IP) basado en formas de onda (el cual es un protocolo de software para radio). El protocolo YAW permite a toda aplicación comunicarse con nodos muy distantes usando paquetes IP. El código de este ejemplo fue

proporcionado por la empresa "THALES Communications & Security" y modificado por A. Nicolás y H. Posadas, en el contexto del proyecto PHARAON ([g)], [aaa)]).

YAW está diseñado por niveles como el modelo OSI. Como forma de onda, implementa los niveles L2 y L3, dependiendo de un ya existente nivel L1. La Figura 100 muestra la estructura de los diferentes componentes que definen el protocolo YAW; en este caso la aplicación está diseñada para comunicaciones vía radio.

Los paquetes YAW son intercambiados usando el protocolo TDMA. En TDMA, la separación de los canales de radio es realizada por medio de la división del tiempo en porciones. En cada una de estas porciones, la estación de radio debe realizar algún tipo de acción (ej. transmitir, recibir, quedarse en suspenso…). De esta forma y dependiendo de la acción implementada en cada porción, se requerirá diferentes tiempos para cada flujo de ejecución.

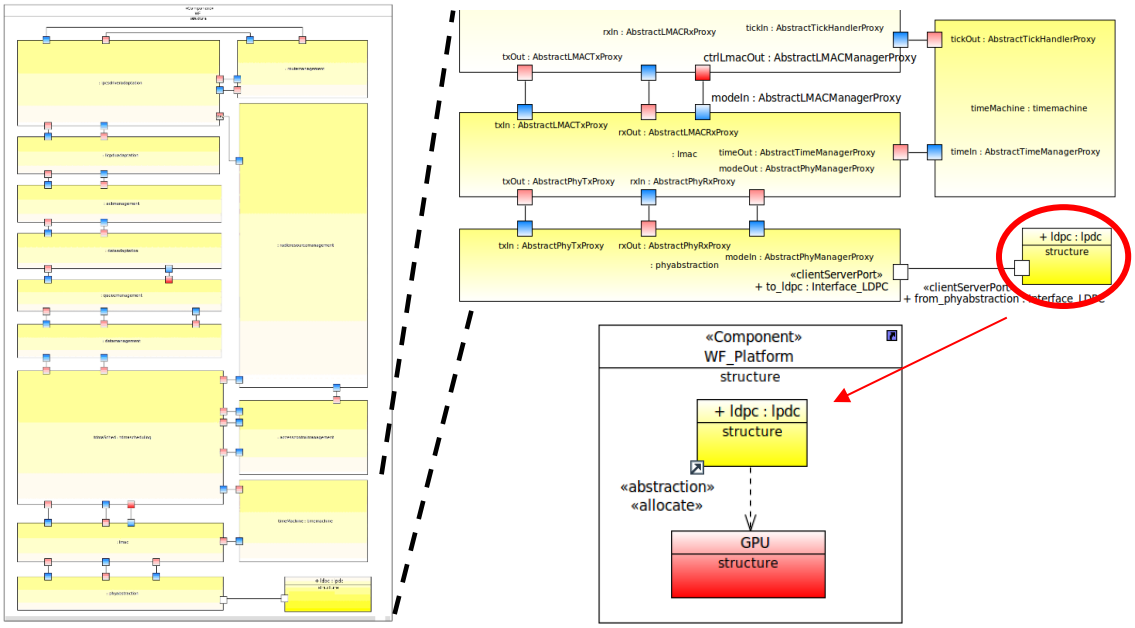


Figura 100 Aplicación YAW y el mapeo a GP-GPU

Como el proceso puede ser asíncrono, es interesante paralelizar la aplicación y analizar el impacto de esa paralización en los diferentes mecanismos de sincronización que usa. También es interesante analizar el balance entre la ganancia de tiempo y el modelo de comunicación inducido cuando se usan varias unidades de procesamiento.

Para analizar todo esto, el ejemplo de aplicación está basado en evaluar diferentes implementaciones del componente "Low-Density Parity Check" (LDPC) (que se muestra en la Figura 100). Estas implementaciones consisten en mapear ese elemento en una CPU y en una GP-GPU de la placa Exynos Octa. Se ha elegido este componente porque es aquel que requiere de más tiempo de cómputo para su ejecución.

Lo primero fue mapear el LDPC a las CPUs con el resto de la aplicación. El segundo experimento fue mapear únicamente el LDPC en la GP-GPU. El último fue mapear el LDPC tanto a la CPU como a la GP-GPU, de esta forma sucesivos flujos concurrentes, que se van creando de forma paralela, no quedan bloqueados, a la espera de un recurso disponible; se van ejecutando en el recurso HW que esté disponible en ese momento, primero en la GP-GPU y luego en la CPU.

Tabla 6 Resultados de la ejecución del componente LDPC sobre diferentes recursos HW

Case	CPU	GPU	CPU+GPU
SMALL	1.22 ms	0.57 ms	0.631 ms
BIG	5.67 s	10.41 s	5.31 s

Los experimentos realizados se muestran en la Tabla 6, donde se puede ver los resultados en el rendimiento dependiendo de diferentes configuraciones. La aplicación fue ejecutada con dos tamaños de dato diferentes, "small" y "big".

Respecto a los resultados obtenidos con el tamaño de datos "small", el mejor rendimiento se obtiene cuando se usa la GP-GPU; el peor rendimiento en las otras dos opciones debido al transferencia de datos adicional. Sin embargo, para el caso "big", el mejor rendimiento es cuando se combinan la CPU y la GP-GPU, siendo la utilización únicamente de la GP-GPU el peor caso.

7. Exploración de Recursos SW

Para mostrar el impacto de los diferentes recursos SW, se utilizó una versión del sistema de visión estereoscópica diferente a la mostrada en la Figura 87 y Figura 97, y que se muestra en la Figura 101. El código de este ejemplo fue proporcionado por la empresa TeDeSys y modificado por A. Nicolás, principalmente, y por mí, en el contexto del proyecto PHARAON ([g)], [aaa)]).

En la figura se puede apreciar como a los canales entre componentes de aplicación se les puede asociar una API concreta o un recurso de sistema operativo para su implementación. Los canales entre los componentes "image"-"starting", "starting"-"stereoMatching" y "stereoMatching"-"triangle" muestran qué recursos se ha de utilizar para su implementación. En el primer caso se utiliza un recurso del sistema operativo (FIFO), y en los otros dos casos se usan dos APIS diferentes (OpenStream y MCAPI, respectivamente).

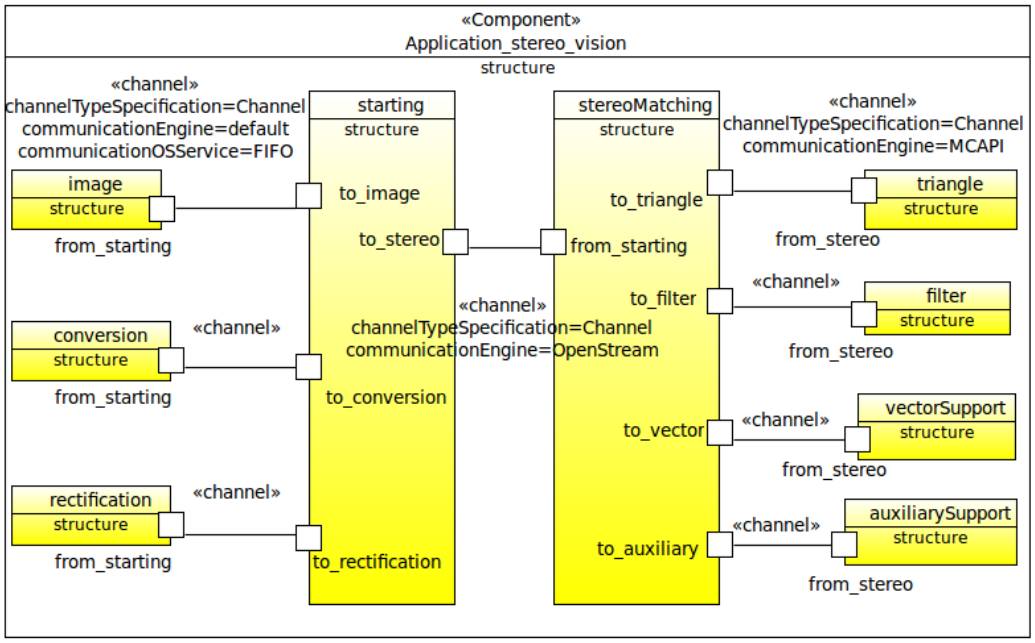


Figura 101 Sistema de Visión estereoscópica, versión 3

Dicho sistema es mapeado a diferentes tipos de plataformas: en nativo, en una Beagle, en una Panda y en una IMX6. Los resultados obtenidos se muestran en la Tabla 7. Para una misma estructura de aplicación y fijada la plataforma, el impacto de los recursos SW usados para la implementación son relevantes a la hora de obtener la implementación final del sistema. Para este experimento se ha considerado que los

componentes de aplicación mostrados en la Figura 101 se han mapeado sobre un único espacio de memoria.

Tabla 7 Resultados del uso de diferentes recursos SW en segundos

Caso	Nativo	Beagle	Panda	IMX6
A. POSIX + Local memory	4.131	26.250	23.217	22.060
B. POSIX FIFOs	4.140	26.290	23.162	22.470
C. POSIX Shared Memory	4.187	28.420	22.878	21.890
D. POSIX Message queue	4.146	26.280	23.012	21.880
E. OpenMP	4.166	24.910	21.883	22.040
F. OpenStream	4.091	24.370	19.884	19.360
G. MCAPI	19.12	234.60	94.768	49.70

De estos experimentos se pueden obtener interesantes conclusiones. Con servicios POSIX, es posible obtener mejores resultados que con memoria local, debido al diferente manejo que se hace de la memoria. Usando OpenMP, los resultados obtenidos sobre las plataformas consideradas son mejores a pesar de la sobrecarga que introduce, mejorando incluso con OpenStream. Los resultados obtenidos con la versión libre de MCAPI son muy pobres. Por tanto, parece que esta API es adecuada para conectar componentes mapeados en diferentes sistemas operativos, pero no para todos los canales del sistema.

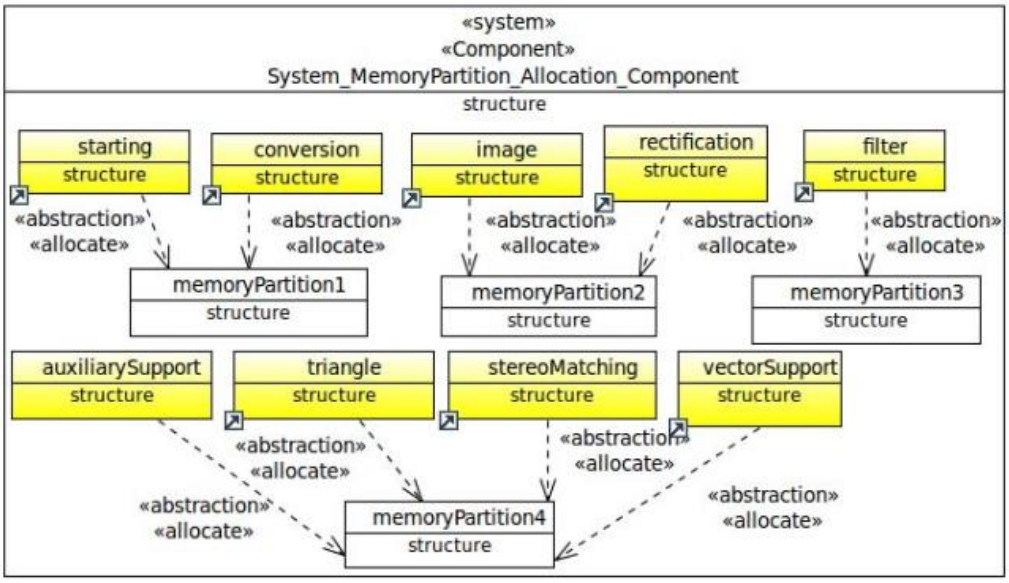


Figura 102 Mapeo a cuatro espacios de memoria.

Un segundo conjunto de experimentos consiste en definir cuatro espacios de memoria donde los componentes de la Figura 101 son mapeados (Figura 102). Los

resultados obtenidos se muestran en la Tabla 8. En este caso, no se han considerados soluciones orientadas a conectar componentes en el mismo proceso.

Tabla 8 Resultados del uso de diferentes recursos SW para un sistema con cuatro espacios de memoria, en segundos

Caso	Nativo	Beagle	Panda	IMX6
I. POSIX FIFO	4.438	26.730	23.587	22.650
J. POSIX Shared Memory	4.858	27.240	25.085	24.410
K. POSIX Message queue	4.322	27.390	23.074	22.480
L. TCP/IP	6.040	27.760	25.243	24.750
M. MCAPI	46.21	696.300	160.928	69.08

Los resultados de la tabla anterior son más lentos que los mostrados en la Tabla 7, debido al incremento en la memoria usada. En el caso de la Tabla 8, un puntero de un parámetro de un servicio necesita de la transmisión del contenido completo del puntero, mientras que en el caso de los experimentos de la Tabla 7, solo se necesita transmitir el puntero.

Finalmente, se realizaron experimentos para un sistema distribuido. La aplicación es ejecutada en dos plataformas, conectadas vía Ethernet. En cada plataforma, lo correspondientes componentes se comunican median el uso de canales FIFO, memoria compartida (SHM) y cola de mensajes (MSQ).

Tabla 9 Resultados para un sistema distribuido en segundos

Caso	Panda-Beagle	Beagle-IMX6	IMX6-Panda	Beagle-Panda	IMX6-Beagle	Panda-IMX6
FIFO-FIFO	26.737	28.730	30.420	26.960	30.020	28.234
FIFO-SHM	26.590	28.280	28.590	26.360	28.400	29.366
FIFO-MQS	26.596	32.050	30.580	26.330	29.500	28.142
SHM -FIFO	26.728	28.110	29.090	26.750	32.850	28.002
SHM- SHM	27.175	30.020	29.650	26.590	34.650	32.056
SHM -MQS	26.654	29.910	32.310	26.530	32.680	28.720
MQS -FIFO	26.575	32.240	29.060	26.740	28.080	28.280
MQS -SHM	26.545	32.120	35.110	27.470	32.360	30.835
MQS- MQS	27.090	31.040	27.700	26.640	29.710	29.852

Mirando los resultados mostrados en la Tabla 9, la mejor solución puede ser llevada a cabo por los mecanismos de comunicación usados para la transmisión de datos internos en cada plataforma. En los resultados mostrados en esa tabla, el espacio de memoria del “filter” (Figura 102) es ejecutado en la segunda plataforma. El componente “filter” requiere de la mayor parte de la capacidad de cómputo. Como consecuencia de

esto, se espera que el mapeo de dicho componente en la plataforma más potente, desde el punto de vista de capacidad de procesamiento, produjera mejores resultados. Sin embargo, los resultados no confirman esta hipótesis; los resultados obtenidos se ven afectados por el mecanismo de comunicación usado para la implementación.

Otro ejemplo en el cual se analizó el impacto de los recursos SW utilizados para la síntesis de las comunicaciones fue con el ejemplo mostrado en la sección *Canales y Concurrencia* de este capítulo, con el ejemplo de la Figura 87.

Los diferentes componentes de aplicación mostrados en la Figura 87 se mapearon a espacios de memoria. La implementación de las comunicaciones entre componentes de diferentes espacios de memoria se realizó mediante la utilización de recursos de sistema operativo, especificados en el modelo como se describió en *Recursos SW* de la sección *Información asociada a la síntesis* del capítulo *Modelado de Sistemas Electrónicos*.

Los resultados de ejecución que se muestran en la Tabla 2 se utilizaron canales FIFO para implementar todos los canales del sistema. En este punto, se van a mostrar el impacto que tiene en el rendimiento la utilización de un tipo u otro de recursos SW de la plataforma para la síntesis de los canales.

Además de esto, se va a realizar este experimento sobre tres diferentes plataformas: una Beagle, la Panda y una i.MX6. Los resultados obtenidos se muestran en la Tabla 10.

Tabla 10 Resultados del uso de diferentes recursos SW para el ejemplo de Figura 87 y Figura 88

Panda	**FIFO**	**Memoria Compartida**	**Cola de Mensajes**
1) sequential	78.6	80.9	80.0
2) parallel	45.1	45.6	44.1
3) pipeline_1	41.5	42.9	42.4
4) pipeline_2	42.1	43.2	43.5
5) pipeline_4	45.4	44.5	45.4
Beagle	**FIFO**	**Memoria Compartida**	**Cola de Mensajes**
1) sequential	225	228	228
2) parallel	227	232	231
3) pipeline_1	228	230	230
4) pipeline_2	228	232	231
5) pipeline_4	229	233	232
i.MX6	**FIFO**	**Memoria Compartida**	**Cola de Mensajes**
1) sequential	79.5	82.1	78.1
2) parallel	44.1	44.0	46.1
3) pipeline_1	42.7	41.2	44.7
4) pipeline_2	25.3	24.0	26.9
5) pipeline_4	24.2	25.7	24.7

Los resultados de la tabla muestran el impacto del mecanismo de implementación seleccionado. Por ejemplo, en la Panda, usar FIFOs es la mejor opción para los casos 1, 3 y 4; la memoria compartida para el caso 5 y la cola de mensajes para el 2. Sin embargo, esto no es verdad en el caso de la Beagle, donde por ejemplo, el caso 2 tiene un mejor rendimiento con FIFOs. En el caso de la i.MX6, la memoria compartida es la mejor opción para los casos 2, 3 y 4.

8. Análisis de Planificabilidad

Como caso de aplicación para realizar un análisis de planificabilidad, se toma como ejemplo una nueva versión de sistema de estéreo visión. El código de este ejemplo fue proporcionado por la empresa TeDeSys y modificado por A. Nicolás, principalmente, y por mí, en el contexto del proyecto PHARAON ([g)], [aaa)]). Para el desarrollo de este ejemplo se tuvo la ayuda de J. Medina y de H. Posadas.

La estructura de la aplicación se muestra en la Figura 103. En este caso, procesamiento de las imágenes se divide en dos fases. En la primera fase, la aplicación está conectada al entorno del sistema para adquirir las imágenes y, una vez procesadas, almacenarlas en él.

El componente "controller" llama a la cámara del entorno para adquirir las nuevas imágenes y las envía a la siguiente componente de la aplicación. El componente "mainLoopHandler" recoge la imagen adquirida por el "controller" y la envía a la segunda fase de la aplicación, el proceso de visión estéreo. Una vez que las imágenes ya han sido procesadas, estas con enviadas al entorno para ser almacenadas.

Esta segunda fase está compuesta por los componentes "StereoMatching", "triangle", "mean", "filter" y "disparity".

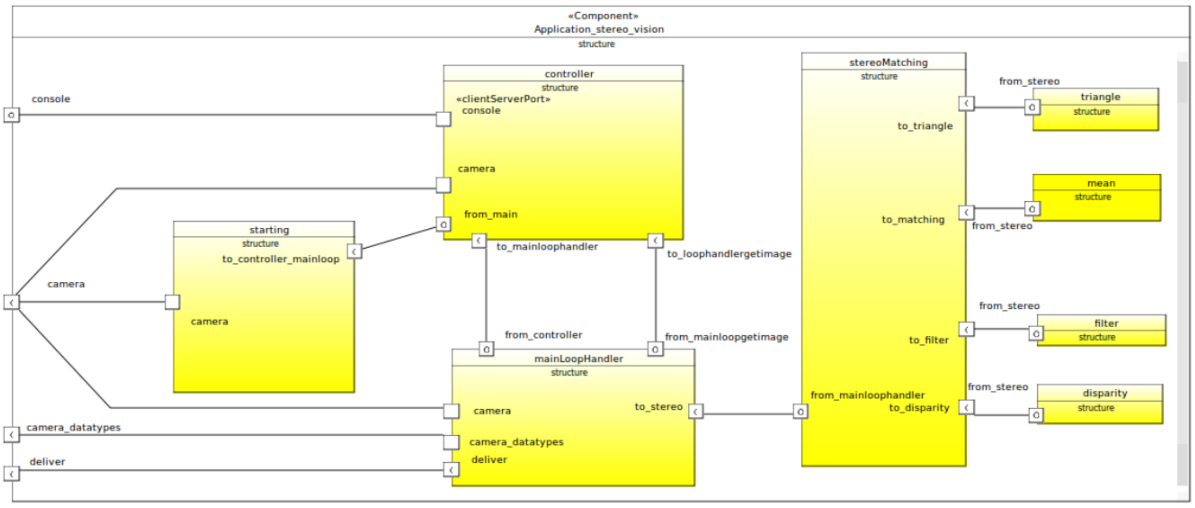


Figura 103 Modelo de estéreo visión, versión 4.

La estructura concurrente está definida en consonancia con las fases de la aplicación anteriormente mencionadas. Se consideran cuatro taras: la tarea uno (“task1”)

adquiere las imágenes (fase 1), mientras que las otras tres tareas (“task2”, “task3” y “task4”) procesan tres pares de imágenes en paralelo.

Respecto a la plataforma HW, se ha considerado para el experimento una placa Intel-Duo. Las tareas “task1” y “task2” son ejecutadas en el procesador 1, mientras que las restantes tareas en el procesador 2.

El objetivo de este caso de uso es garantizar una cierta tasa de procesado de imágenes, analizando el tiempo disponible que puede ser utilizado para las actividades no-críticas. En este contexto, el primer paso es estimar los tiempos de ejecución y realizar el análisis de planificabilidad.

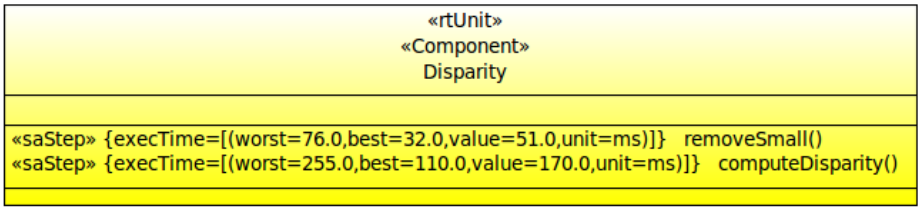


Figura 104 Componente de aplicación y sus funciones asociadas con sus tiempos de ejecución.

Para realizar el análisis estático, lo primero es caracterizar las funciones con sus tiempos de ejecución, como se muestra en la Figura 104. Luego se define el escenario de análisis (Figura 105), donde se especifica que tarea lo va a hacer y que funciones se van a ejecutar.

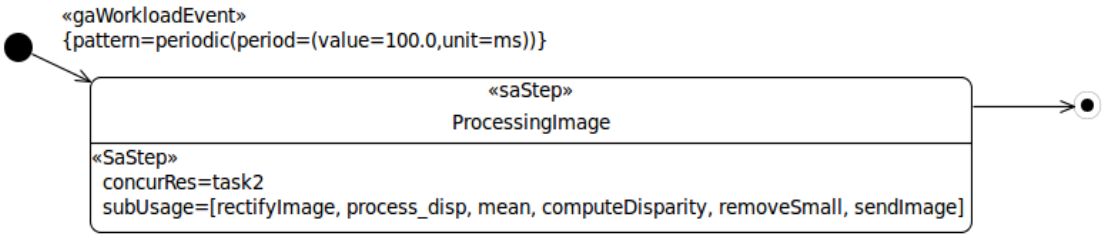


Figura 105 Definición del escenario de análisis

Las funciones consideradas en el análisis se muestran en la Tabla 11, donde el nombre y el MOET (“Maximum Observable Execution Time”) son anotados, dependiendo de modo de operación del procesador, definidos como el mejor, el peor y el medio.

Tabla 11 Funciones y sus tiempos de ejecución

Función	MOET 3.8GHZ	MOET 2.4GHZ	MOET 1.6GHz
main_app	30	35	50
rectifyImage	11000	18000	28000
Process_disp	55000	88000	130000
computeDisparity	110000	170000	255000
removeSmall	28000	42000	62000
mean	31000	49000	74000
sendImage	500	800	1000

Lo siguiente a hacer es el análisis de planificabilidad con MAST, considerando una serie de reglas. Primero, se establece un margen de seguridad en la utilización del procesador: esta debe ser de menos del 90%. Segundo, se considera un 5% adicional de utilización del procesador como consecuencia del sistema operativo.

Los resultados proporcionados por MAST se muestran en la Tabla 12, capturando la utilización de cada procesador en fps. Debido a las restricciones anteriores, los casos de “fps=12” y “fps=16” son considerados inválidos.

Tabla 12 Utilización de los procesadores de la plataforma

Procesador	Utilización (8fps)	Utilización (10fps)	Utilización (12fps)	Utilización (16fps)
Procesador 1	33.4%	41.47%	49.9%	66.3%
Procesador 2	66.7%	81.4%	99.3%	133.5%

9. Exploración del Espacio de Diseño

En *Exploración Automática* de la sección *Análisis: Exploración del Espacio de Diseño* del capítulo *Entorno de Desarrollo: Integración* de *Herramientas* se explicó que, a partir del modelo, se genera, de forma automática, toda la información necesaria para poder conectarse con el conjunto de herramientas que permitían la simulación funcional del sistema (SCoPE y VIPPE), y el proceso de exploración automática (realizado por MOST). De esta forma se podría encontrar aquellas configuraciones del sistema (en términos de valores de atributos de los recursos HW de la plataforma, así como del mapeo) que nos diera el mejor resultado en función de las métricas definidas. El objetivo de este proceso es encontrar la mejor configuración del sistema ya que este debe cumplir todas las restricciones y optimizar las métricas seleccionadas.; en el caso concreto de este ejemplo es que el consumo de potencia sea menor que 2 vatios. Como caso de uso, el entorno de diseño se aplicó para obtener la mejor configuración de la plataforma y del mapeo funcional, a un EFR vocoder.

El código fue desarrollado por F. Herrera y el ejemplo se realizó con la colaboración entre F. Herrera, principalmente, H. Posadas, G. Palermo, en el contexto del proyecto COMPLEX ([xx)]).

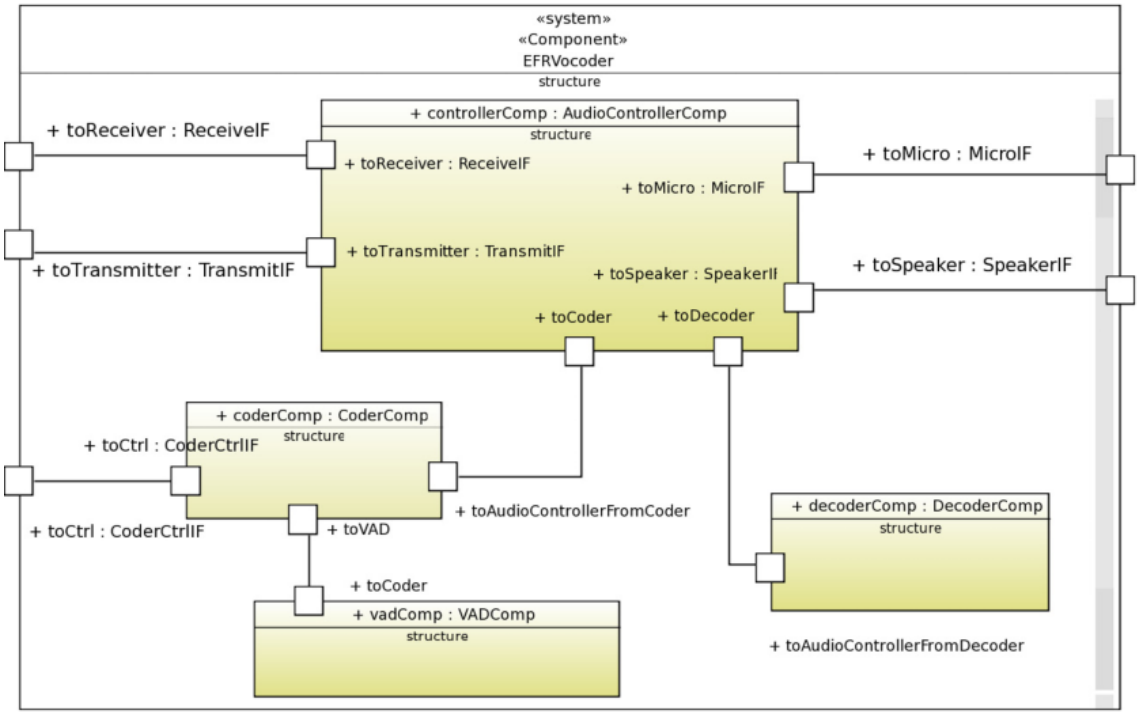


Figura 106. Ejemplo de EFR vocoder

El modelo de aplicación del EFR (“Enhanced Full Rate”) vocoder se muestra en la Figura 106. El vocoder EFR es un sistema de compresión/descompresión ampliamente usado en las comunicaciones móviles para ahorrar ancho de banda.

El vocoder está compuesto de una rama de codificación y una de decodificación. El codificador se encarga de transformar el audio en un formato comprimido, de acuerdo a un modelo de producción de voz humana. La rama de decodificación implementa la operación inversa.

La funcionalidad es capturada mediante cuatro componentes como muestra la Figura 106: “AudioController”, “Coder”, “Decoder” y el “VoiceActivity”. Estos componentes son mapeados a la plataforma como se muestra en la Figura 107, donde se puede ver el número de procesadores.

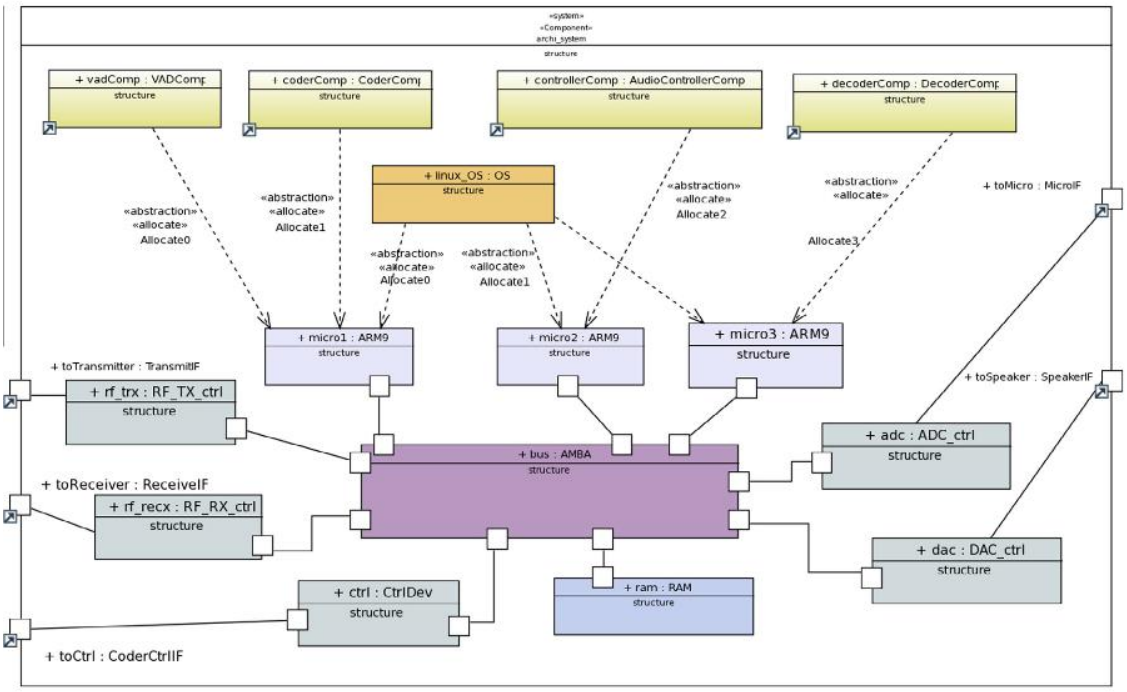


Figura 107. Arquitectura y mapeo del de EFR vocoder.

A partir de aquí, se modela el espacio de diseño para su exploración. La Figura 107 muestra un mapeo de la funcionalidad que es, en principio, fijo. Sin embargo, se puede especificar una variable de diseño que modifique este mapeo. Esto se muestra en la Figura 108, donde se especifican todas las combinaciones de potenciales mapeos de los componentes de aplicación en los recursos de la plataforma, que se pueden dar.

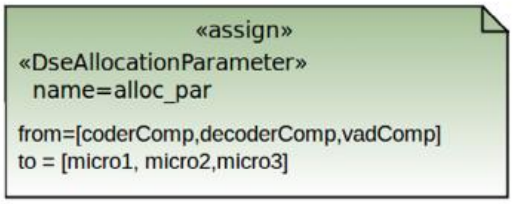


Figura 108. Especificación de potenciales mapeos.

Más concretamente, la Figura 108 modela 27 posibles mapeos (resultado de combinar cada componente en cada procesador): este combinación viene dado de considerar el mapeo de "coderComp" en cualquier de los tres procesadores presentes en Figura 107, y de la misma manera para el resto de los componentes de aplicación.

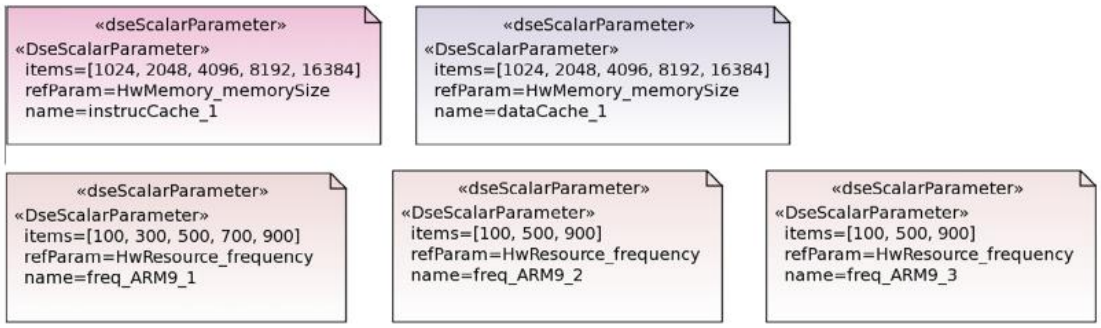


Figura 109. Ejemplo de Variables de exploración de los componentes HW

La Figura 109 muestra las variables de exploración de la plataforma HW del sistema. En este ejemplo, este espacio está definido por los potenciales valores de los tamaños de las caches de datos e instrucciones de los procesadores (1K, 2K, 4K, 8K y 16K), y por considerar diferentes frecuencias en dichos procesadores.

Por tanto, el tamaño del espacio de diseño considerado en función de los valores que pueden tomar las variables de diseño son: (5 frecuencias de los procesadores) x (5 tamaños de la cache de datos) x (5 tamaños de la cache de instrucciones) = 125. Este número es únicamente para un solo procesador.

Este número de combinaciones posibles pueden ser reducidas estableciendo restricciones entre las variables de diseño. Por ejemplo, la Figura 76 muestra cómo se reducen las potenciales mapeos de la aplicación que se muestran en la Figura 108, reduciendo los potenciales mapeos de 27 a 18.

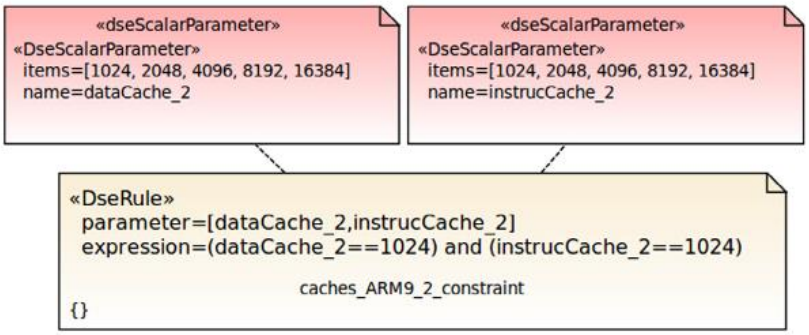


Figura 110. Regla de diseño para elementos HW

La Figura 110 muestra cómo se reducen los potenciales valores de la cache de datos y de instrucciones especificación una relación entre ellas, mediante una regla de diseño.

De la misma forma, para hacer este experimento se consideró el entorno del sistema, en forma de generador de estímulos. Este se mostró en *Modelo de Estímulos* de la sección *Modelado del Entorno* del capítulo *Modelado de Sistemas Electrónicos*.

9.1 Resultados Experimentales

Con las variables de diseño anteriormente mostradas se comienza el proceso de exploración del espacio de diseño. Se decidió fijar el mapeo de algunos de los componentes de aplicación para evitar la exploración de puntos de diseño equivalentes y para explorar las frecuencias de trabajo de los procesadores de forma independiente. La razón para esto fue que, en un análisis inicial, se observó que la carga computacional de los componentes no estaba balanceada, que el codificador requería mucho más fuerza de cómputo que el decodificador.

El proceso de exploración cubrió diferentes mapeos arquitecturales considerando los cuatro componentes de aplicación de vocoder (Figura 106) y los tres procesadores (Figura 107) presentes en la plataforma. Como se mencionó con anterioridad, el mapeo de los componentes del codificadoor y el decodificar se fijo en el primer procesador, explorando los efectos de los portenciales mapeos de los otro dos componentes de aplicación en el resto de procesadores. Esto se hace mediante la definición de dos reglas de diseño aplicadas a la Figura 108.

Como consecuencia de fijar dichos mapeos, el procesador 1 tendrá un alta carga de cómputo. Por esto, cinco potenciales frecuencias y cinco tamaños de cache han sido explorados para este procesador. Para lo otros procesadores, fijadas los maños de las

caches por la Figura 110, se han explorado tres frecuencias para cada uno de ellos. Por tanto, el número de soluciones ha explorar son:

N = 5(freq.proc1) x 5(tamaño de caches de proc1) x 3(freq.proc2) x 3(freq.proc3) x 6 (mapeos) = 1350 soluciones a explorar.

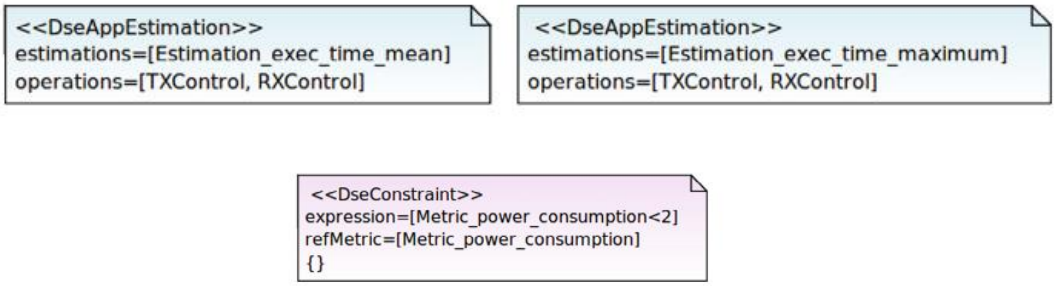


Figura 111. Métricas y funciones a comparar.

Todo el proceso de automatización de exploración lo realiza la herramienta MOST. Esta devuelve un detallado análisis de las métricas consideradas en el proceso de exploración. En este caso, se ha analizado el máximo y el tiempo medio necesitado para las funciones de recibir y trasmitir datos, comparándolas y relacionándolas con el número de instrucciones ejecutadas y el consumo de potencia, acorde con las diferentes configuraciones definidas en el espacio de diseño. Esto se captura en la Figura 111 (imágenes reportadas por MOST).

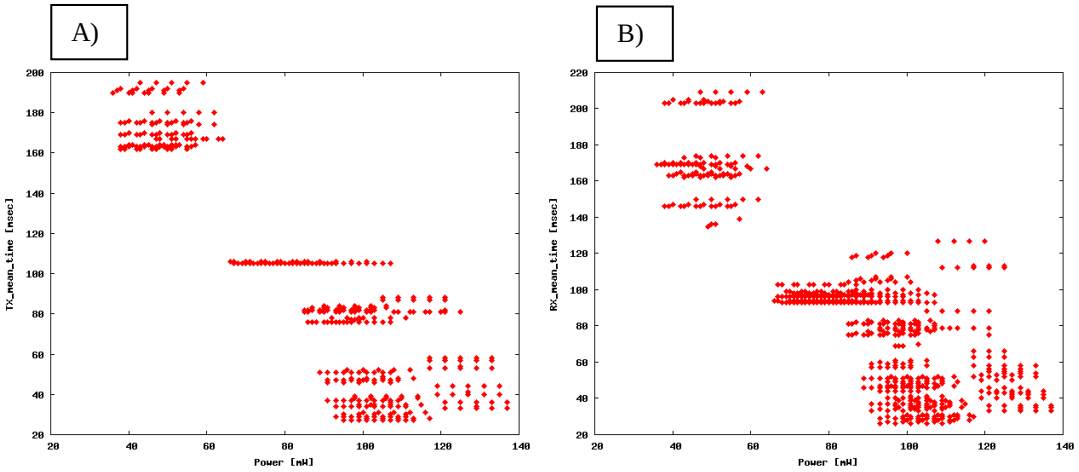


Figura 112 Exploración de los resultados: A) "Power-TX_mean_time" y en b) "Power-RX_mean_time"

La Figura 112 (imágenes reportadas por MOST), proporcionada automáticamente por la herramienta MOST, muestra el resultado del consumo para la transmisión ("Power-TX_mean_time") y para la recepción ("Power-RX_mean_time"), respectivamente. Mirando la figura, se puede ver una clara relación entre los tiempos de recepción y transmisión y los consumos. En ambos casos, las soluciones están situadas en dos grupos: uno en la parte superior izquierda caracterizado por un bajo consumo de potencia y una alta latencia, y otro en la parte inferior derecha caracterizado por un alto consumo y una baja latencia.

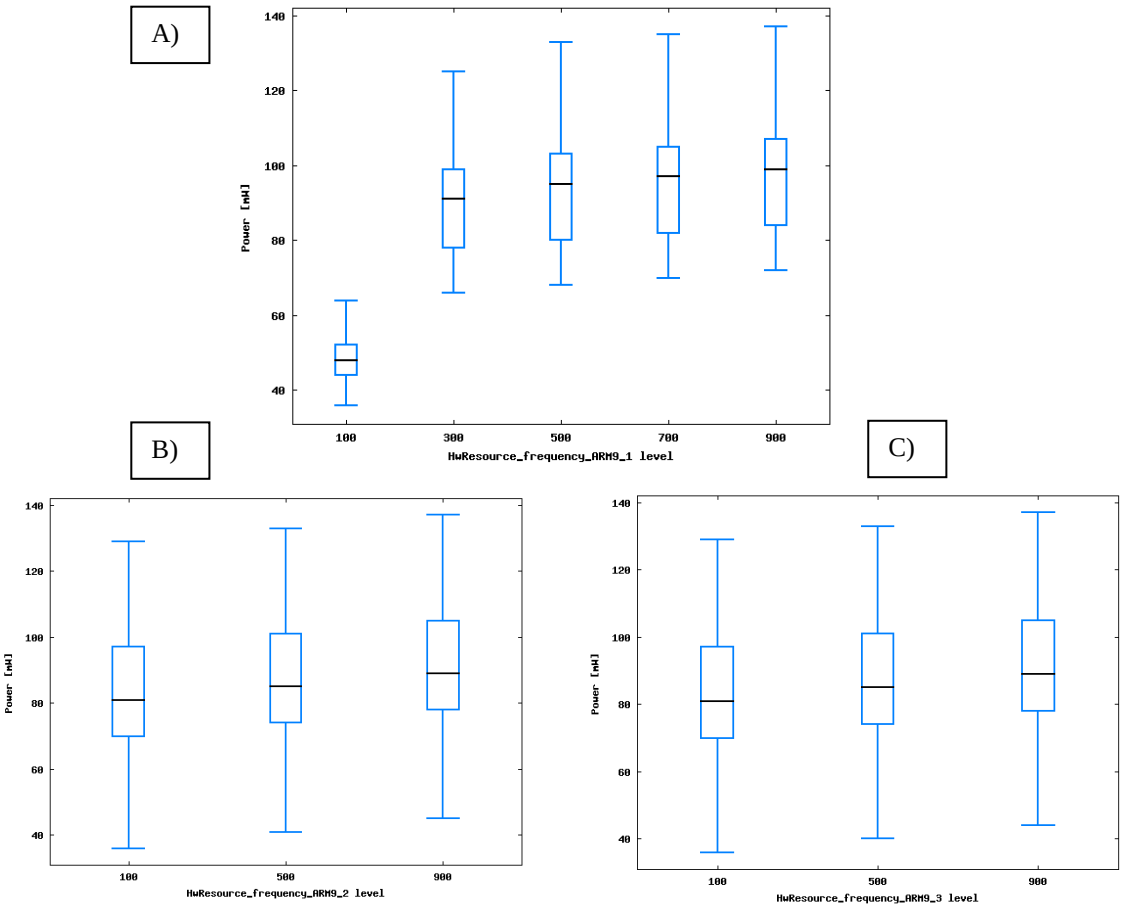


Figura 113 Resultados obtenidos después del proceso de exploración

Las gráficas de la Figura 113 (imágenes reportadas por MOST) que muestra el análisis del consumo de potencia en función de los posibles valores de frecuencia para cada procesador considerado en la plataforma destino (según los valores capturados en Figura 109 y Figura 110 y el mapeo de funcionalidad a procesador capturada en Figura 76). Los resultados muestran que la tendencia y la sensibilidad en el consumo de potencia al aumentar la frecuencia es similar para los procesadores "micro2" y "micro3" (Figura

113b y la Figura 113c). Sin embargo, cuando se trata del procesador "micro1" (Figura 113a), el incremento en el consumo de potencia en el tránsito desde la frecuencia más baja (100 MHz) hasta la siguiente frecuencia (300MHz) es significativo, mientras sucesivos incrementos de la frecuencia producen un ligero aumento en el consumo. Esto es consistente con la mayor cantidad de funcionalidad asignada al "micro1".

La misma comparativa para los tiempos máximo y medio es mostrada en la Figura 114 (imágenes reportadas por MOST) en relación a las frecuencias del procesador 1. Ya que la reducción de la frecuencia afecta a los tiempos de computación. De esta forma, estos datos pueden ayudar al diseñador a tomar la mejor solución a su problema.

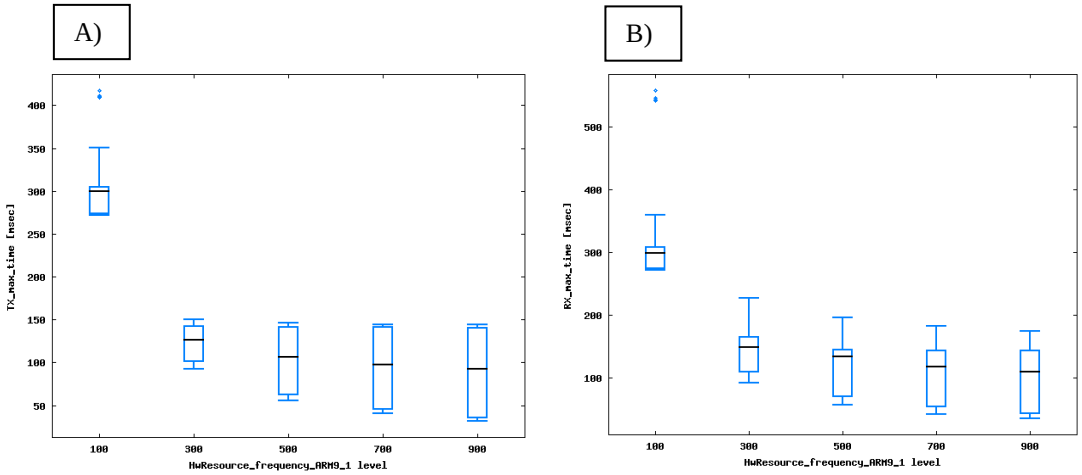


Figura 114 Resultados obtenidos para: a) TX_max_time y b) RX_max_time para las potenciales frecuencias del procesador 1

Mirando los parámetros de mapeo, la Figura 115 (imagen reportada por MOST) muestra como el mapeo del componente del decodificador o sobre el procesador 1 o sobre el procesador 2 no cambia el comportamiento del sistema, pero sí la distribución del tiempo máximo de recepción (y también el tiempo medio) a lo largo de las diferentes configuraciones. De hecho, mapear el decodificador en el procesador 1 parece ser más robusto que mapearlo al procesador 2, ya que la mayoría de los valores para el tiempo máximo son más bajos que la media. El valor máximo es más pequeño que el máximo para el mapeo sobre el procesador 2.

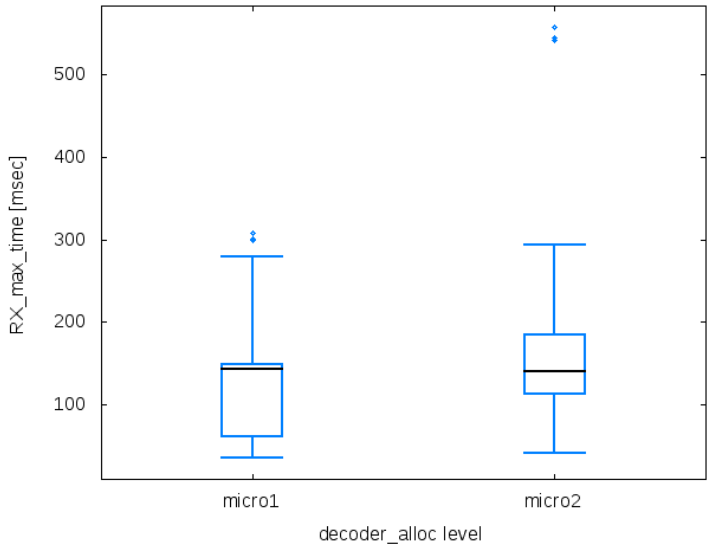


Figura 115 Resultados obtenidos para el RX_max_time en función de los mapeos

Finamente, la Figura 116 (imagen reportada por MOST) muestra el impacto de los tamaños de la cache sobre el tiempo máximo de transmisión. Analizando la figura es posible ver que las caches más pequeñas que 2KB producen un dramático incremento en los tiempos de ejecución, mientras que las caches mayores a 4KB casi no tienen impacto en el resultado.

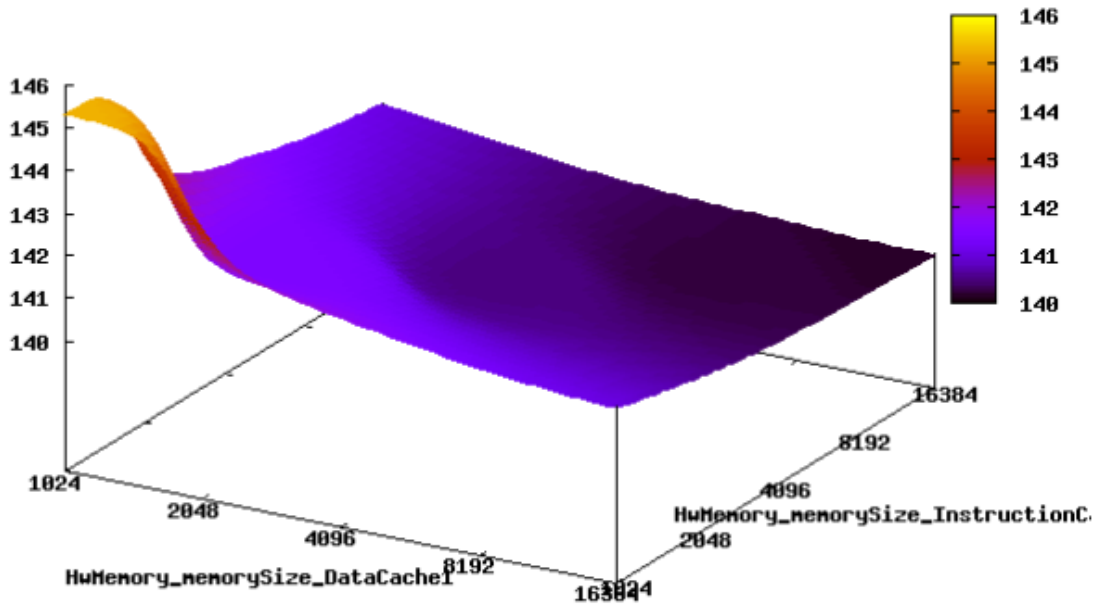


Figura 116 Resultados obtenidos para el TX_max_time en función de los tamaños de la caches

10. Redes de Sensores

Como caso de uso de redes de sensores, para mostrar toda potencia del modelo UML/MARTE y el conjunto de herramientas implementado se pensó en explorar la tipología y el tipo de nodos de una WSN, centrándose en la obtención, como métrica de exploración, el consumo de energía de los nodos al verse afectados por diferentes tipos de ataques, en diferentes configuraciones de red.

La consideración del consumo de energía como parámetro a analizar se debe a que suele ser la principal limitación de los nodos en las WSN, ya que uno de los requisitos principales de esto elementos es su necesidad de consumir poca energía. Los nodos son dispositivos que obtienen su energía de baterías; por tanto, la vida útil de un nodo depende, directamente, de la duración de la batería.

Unos de los efectos más inmediatos de la acción de los atacantes sobre los nodos es el aumento de consumo de energía, consecuencia de las disfuncionalidades introducidas por dichos ataques en el correcto funcionamiento del nodo.

El ejemplo concreto es que la red de nodos monitoriza una propiedad física de un lugar (por ejemplo la temperatura). En el ejemplo, la red de sensores adquiere información periódicamente cada 3 segundos. Estos datos son recogidos por un nodo central que los procesa. De acuerdo con los resultados obtenidos durante este procesamiento, el nodo central toma una decisión sobre la integridad de la instalación a controlar. Este ejemplo fue el resultado de la colaboración de H. Posadas, A. Díaz, en el contexto del proyecto TOISE [yy)].

Para explorar la configuración de la red, se consideran tres variables: los tipos de nodos utilizados para crear la red, la topología de red y la configuración de estos nodos. Los tipos de nodos se definen por su estructura interna HW/SW. La complejidad de su estructura HW/SW (número de procesador, tipo de batería, etc.) tiene un gran impacto en el diseño de WSN dependiendo del presupuesto disponible para comprar los nodos.

Las redes se construyen en bases a dos tipos diferentes de nodos: “Controller” y nodos de sensores (sección *Vistas: Modelo Específico de Plataforma para Sistemas Distribuidos* del capítulo *Modelado de Sistemas Electrónicos*, figuras Figura 17 y Figura 18). El nodo “Controller” es responsable de coordinar y controlar la red y comunicarse con otros sistemas externos a la red. El nodo sensor tiene un sensor para leer la temperatura ambiental. Cuando los nodos terminan de realizar su funcionalidad asignada,

cambian a un modo de reposo para reducir el consumo de energía y aumentar la duración de la batería.

Cada tipo de nodo ("Controller" y Sensor) ejecuta un software diferente en el mismo sistema operativo, FreeRTOS. La funcionalidad básica de cada tipo de nodo se describe brevemente:

- "Controller": este nodo es responsable de recibir mensajes de los nodos sensor y transmitirlos a una red externa. Cuando todos los nodos sensor están despiertos, el nodo espera los datos sobre la temperatura ambiental de cada uno de ellos. Cuando este nodo tiene la información de todos los nodos sensor, compone un mensaje y lo envía a través de un módulo GPRS. Después de esto, el nodo se duerme durante 3 segundos. Su estructura interna se muestra en la Figura 17.
- Sensor: después de despertar, lee el sensor y envía los datos al "Controller" o a otro nodo que pueda alcanzar el nodo "Controller". Cuando termina su función, duerme durante 3 segundos. Por lo tanto, la frecuencia de adquisición de datos es cada 3 segundos. Su estructura interna se muestra en la Figura 18.

Además del tipo de nodo, también se consideraron otro tipo de propiedades físicas que tienen impacto en la probabilidad de éxito de la comunicación entre nodos. Estas propiedades son modeladas teniendo en cuenta la probabilidad de éxito de comunicación (modelado por el atributo "probability", *Redes de sensores* de la sección *Sistemas Distribuidos* del capítulo *Modelado de Sistemas Electrónicos*). El nivel de seguridad de la instalación tiene que ser considerado también en el modelado de la red. Para ello, se consideran los diferentes valores "transmPower", "numOfRetries" y "encrypted" (*Redes de sensores* de la sección anteriormente citada) para especificar la comunicación del nodo.

10.1 Explorando la topología de la red

El primer experimento consiste en evaluar la configuración de red que se muestran en las figuras Figura 117, Figura 118 y Figura 119.

La Figura 117 muestra una red con tipología lineal, donde cada nodo transmite la información que él ha capturado, acumulando la información adquirida por los otros nodos que han sido situados en una posición anterior a la suya, en la cadena de nodos.

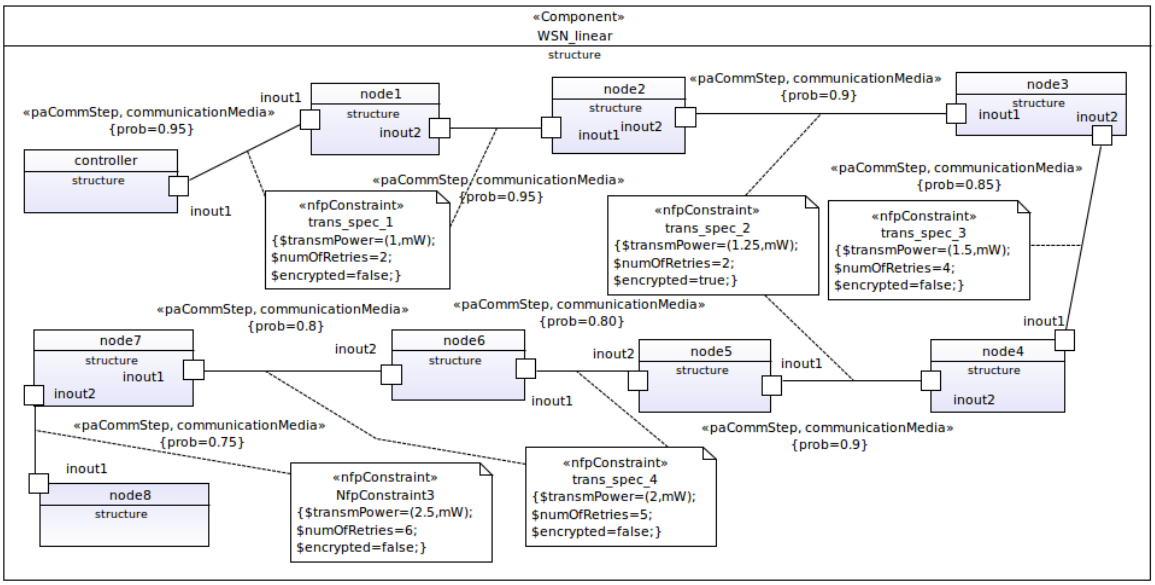


Figura 117 Topología Lineal

La Figura 118 muestra una topología de estrella, donde las estaciones están conectadas directamente con el nodo central, y todas las comunicaciones deben necesariamente ir a través de este nodo.

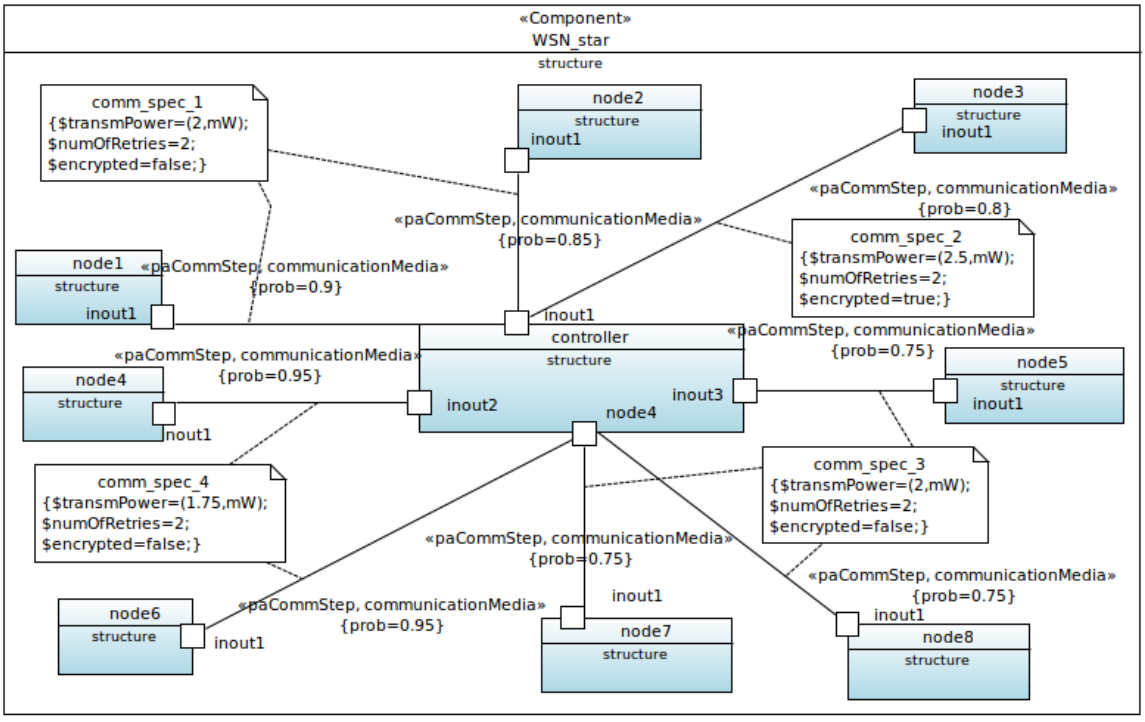


Figura 118 Topología Estrella

También se pueden capturar topologías de red de forma irregular, como se puede ver en la Figura 119.

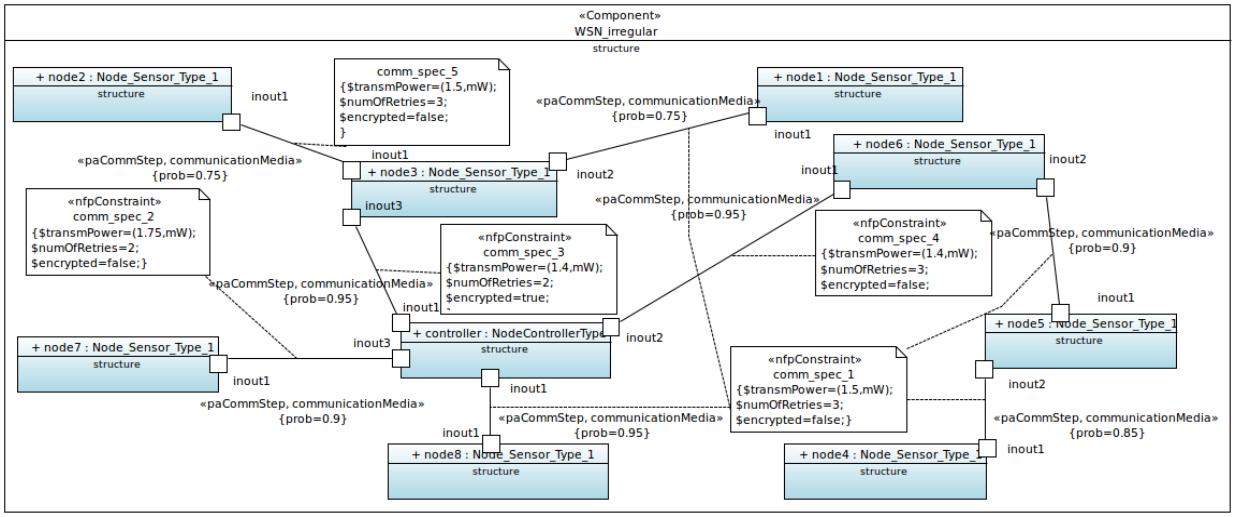


Figura 119 Topología irregular

Los nodos sensor se ha identificado en las anteriores figuras como "node$_i$" con i = 0...8.

10.2 Explorando tipos de ataques

Ahora se han considerado diferentes tipos de ataques, para que afecten al funcionamiento normal de nodos de la red. Los siguientes experimentos consisten en aplicar diferentes tipos de ataques a la red.

Algunos nodos han sido seleccionados para sufrir ataques dependiendo de su papel en la red. Por ejemplo, el "node3" genera información importante que tiene un alto impacto en el nivel de seguridad de la red, por lo tanto, todas sus comunicaciones están cifradas (“encrypted” = true), aumentando su consumo de energía debido al proceso de cifrado. Por lo tanto, el "node3" es atacado debido a la importancia de la información que este nodo tiene que enviar.

Para ello, el atacante seleccionado para ser aplicado a este nodo es del tipo “Collision” (*Ataques a Redes*, sección *Modelado del Entorno* del capítulo *Modelado de Sistemas Electrónicos*) ya que pueden provocar que el nodo no envíe sus datos, afectando a la integridad de la red.

El siguiente nodo en ser atacado es el "node7". En este caso, este nodo es atacado por una "Hello Flood" (*Ataques a Redes*, sección *Modelado del Entorno* del capítulo *Modelado de Sistemas Electrónicos*), que intenta consumir toda la energía en la batería del nodo. Este hecho es importante ya que el acceso a este nodo para cambiar la batería es muy complicado en términos de tiempo y dinero requerido, debido a su localización geográfica.

Finalmente, el último nodo atacado es "node6". Este nodo sufre un ataque de "Looping in the Network" (*Ataques a Redes*, sección *Modelado del Entorno* del capítulo *Modelado de Sistemas Electrónicos*). Este nodo es un nodo raíz usado para transmitir la información de otros nodos (redes Figura 117 y Figura 119). De esta manera, este nodo se vuelve inútil y, por lo tanto, otros nodos se vuelven indirectamente inútiles también.

Para cada topología de red simulada (Figura 117, Figura 118 y Figura 119) se modelan tres atacantes. Para ser consistentes en los experimentos cada topología de red tiene el mismo tipo de atacantes asociados: ruido, inyector de paquetes y mixto. Cada tipo de atacante tiene los mismos valores para sus atributos (por ejemplo, los atributos para modelar un ataque de ruido deben tener los mismos valores para “type”, “errorRate” y “activeTime”).

El modelado del ataque “Collision” se muestra en la Figura 120, para cada tipo de topología de red considerada en el ejemplo. Este tipo de ataque afecta a los enlaces entre nodos. En este caso, el ataque de colisión tiene que identificar el enlace que el "node3" utiliza para transmitir sus datos. En el caso de la Figura 117, este enlace está entre el "node3" y el "node2", identificado en la Figura 120a como "conn_node3_node2". En los casos de la Figura 118 y la Figura 119, el enlace se identifica como "conn_node3_controller_star" y "conn_node3_controller_irregular", respectivamente (Figura 120b y Figura 120c).

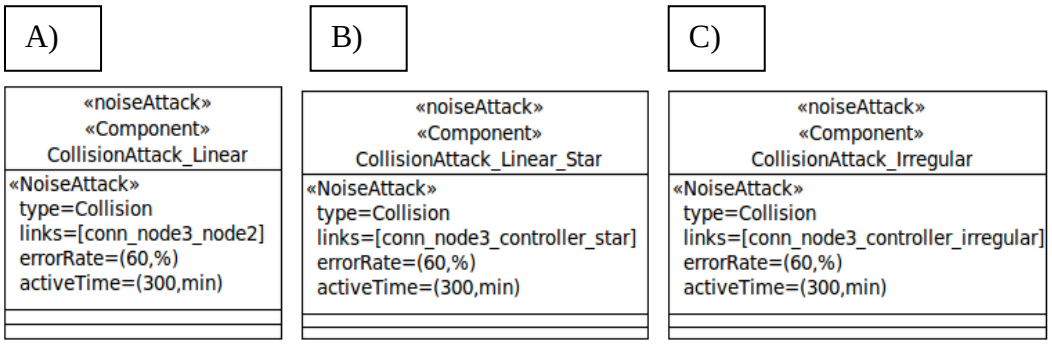


Figura 120 Modelado del ataque “Collision” para el “node3” en cada topología de red

El ataque “Hello Flood” es del tipo inyector de paquetes y ataca el "node7". La Figura 121 muestra los atributos que caracterizan el ataque. En este caso, los tres modelos de ataque de “Hello Flood” tienen la misma representación.

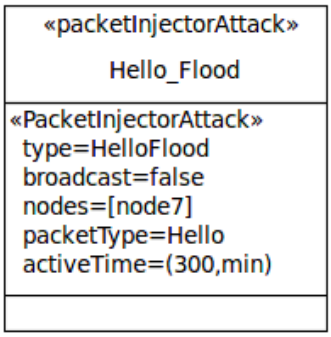


Figura 121 Modelado del ataque “Hello Flood” para el “node7”

Finalmente, el ataque “Looping” es del tipo mixto y ataca el "node6" y el enlace que el nodo utiliza para la transmisión de sus datos. En la red de la Figura 117, el enlace afectado es aquel que conecta el "node6" y el "node5", designado en la Figura 122a como "conn_node6_node5".

En las redes de la Figura 118 y la Figura 119, el enlace afectado es la conexión entre el "node6" y el "controller". Estos se muestran como "conn_node6_controller_star" y "conn_node6_controller_irregular" en la Figura 122b y la Figura 122c, respectivamente.

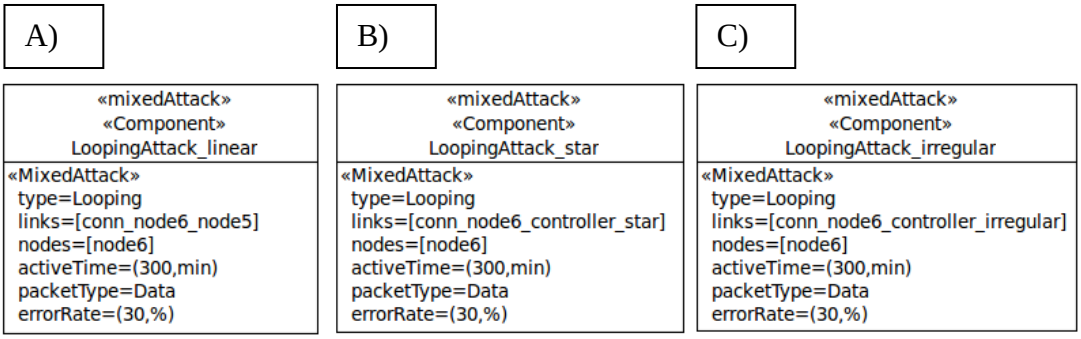


Figura 122 Modelado del ataque “Looping” para el “node6”

10.3 Resultados de la simulación

Las tres topologías de red fueron simuladas cuando no hay ataques y cuando se consideran los tres tipos de ataques descritos en la sección anterior. Los resultados de simulación se han obtenido mediante el simulador de redes, basado en SCoPE, y que se presentó en [87]; en el trabajo [a)] también se puede ver una descripción del mismo.

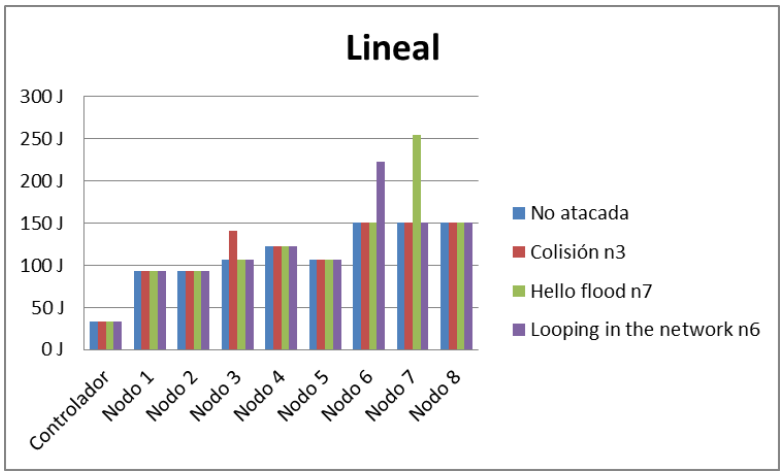


Figura 123 Resultados de consumo de la red lineal

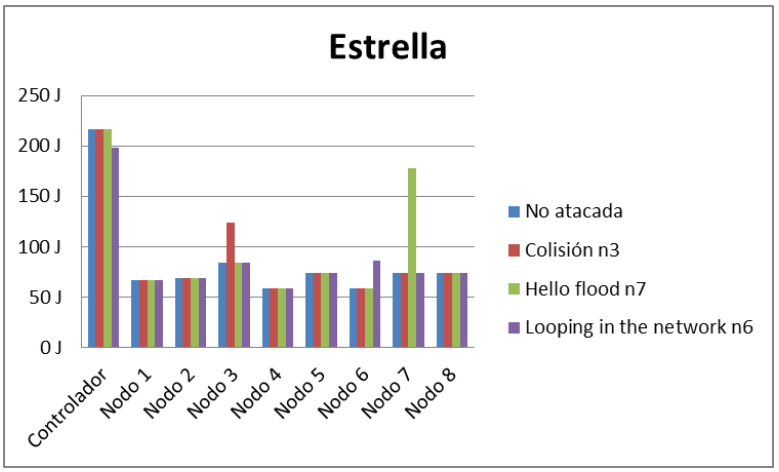


Figura 124 Resultados de consumo cuando de la red en estrella

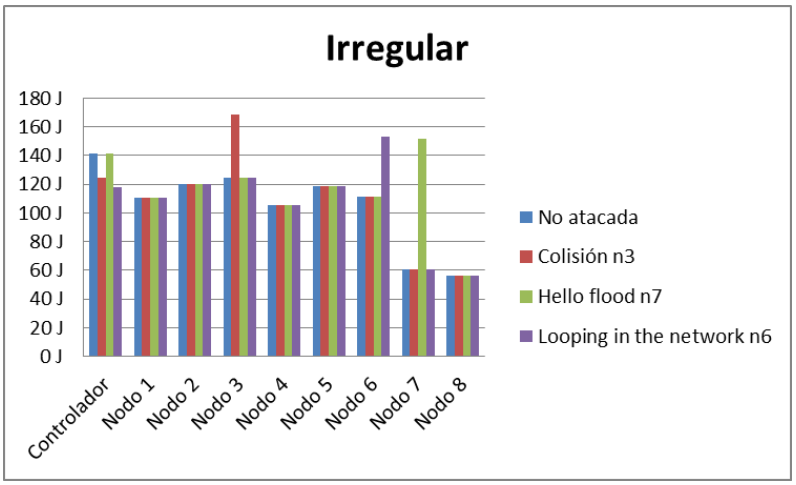


Figura 125 Resultados de consumo cuando de la red con estructura irregular

Como puede observarse en las figuras Figura 123, Figura 124 y Figura 125 (proporcionadas por A. Díaz), el consumo del nodo "controller" en la red lineal es menor que las otras redes, ya que recibe un número menor de paquetes. Sin embargo, en la red lineal, el consumo de los nodos es mayor debido al aumento de la carga de trabajo y el número de paquetes transmitidos. También se puede observar que la más balanceada en lo que al consumo se refiere es la irregular.

Ahora se va a analizar el efecto de los tres ataques previamente modelados sobre las diferentes configuraciones de red. Las figuras Figura 123, Figura 124 y Figura 125 muestran la variación del consumo de cada nodo en las diferentes configuraciones como consecuencia de los ataques. En lo referente al ataque "Collision", cuando se comparan el impacto de los diferentes ataques, se puede ver que la red en estrella se ve más afectada que las otras configuraciones de red. Se puede apreciar que se ha reducido el consumo del "controller" en la red irregular durante este ataque. Esto se debe a la disminución en el número de paquetes recibidos por él (aumentando el proceso de toma de decisiones). Esta reducción en la recepción de paquetes se debe a que la configuración del "node3" sólo permite dos reintentos para enviar paquetes, por lo que se pierde un gran número de paquetes. En otros casos, el número de reintentos para enviar la información es mayor, lo que lleva una reducción en la tasa de pérdida de paquetes.

Respecto al ataque "Hello Flood", en las tres gráficas se puede ver que este ataque implica un gran aumento en el consumo del nodo atacado ("node7").Este aumento en el consumo implica que el ciclo de vida de ese nodo se reduce a la mitad, lo que implica la necesidad, en poco tiempo, de la sustitución de la batería del nodo, con los costes asociados con esta acción.

Finalmente, el ataque “Looping” afecta a las tres redes por igual. Sin embargo, en el caso de la red irregular y la red en estrella, el “controller” pierde algunos paquetes, provocando tomas de decisiones incorrectas. En la configuración de red lineal, el “controller” recibe el mismo número de paquetes, sin variación en su consumo. Sin embargo, la información en estos paquetes está dañada; la información de algunos nodos no está incluida en el proceso de toma de decisiones.

Para más tipos de análisis sobre este caso de uso, ver [a)].

Capítulo VIII

Conclusiones

Como se ha ido mostrando a lo largo de esta tesis, todo el proceso de especificación, diseño, validación e implementación necesarios en el desarrollo de sistemas empotrados puede resultar una tarea ardua, compleja y con múltiples variantes. A lo largo de los diferentes casos de uso que se han ido mostrando este documento, se ha podido ver cómo, durante el proceso de diseño, se pueden considerar múltiples opciones de diseño, incluyendo diferentes alternativas en la estructura concurrente, mapeos, recursos de plataforma usados o propiedades de estos recursos, como la potencia de emisión de un nodo en una red. Esta cantidad de alternativas, por sí mismo, supone una ingente cantidad de información que hay que manejar en el proceso de diseño, implicando diferentes áreas de trabajo cada una de ellas con sus propias particularidades. Además, aunque cada una de ellas se puede abordar por separado, es necesario no perder la perspectiva de la completitud de las partes individuales con respecto al sistema global, con el fin de evitar potenciales problemas e inconsistencias cuando se quiera integrarlo todo.

Para proporcionar una solución factible a esta tarea de diseño integral, esta tesis propone el uso de modelos de alto nivel basados en UML/MARTE como elemento central de todo el proceso de desarrollo del sistema empotrado. Como se ha ido viendo con los diferentes ejemplos, estos lenguajes presentan una gran versatilidad expresiva, muy potente para caracterizar sistemas empotrados y que abarca desde la especificación de requisitos funcionales-no funcionales, hasta el modelado de propia estructura (aplicación y plataforma) del sistema, así como para la descripción de escenarios de análisis y el enlace con las herramientas necesarias para realizar todo el proceso de diseño.

Esta potencia expresiva viene dada por el gran número de elementos disponibles, con un sinfín de recursos y reglas gramaticales que proporcionan tanto UML como MARTE. Este hecho hace que los documentos donde se especifican y describen estos mecanismos expresivos sean de gran tamaño.

En esta línea, la primera conclusión obtenida tras el trabajo realizado viene en relación al propio uso de UML/MARTE. A lo largo del tiempo en el que se ha realizado el desarrollo de esta tesis, ha sido muy común lidiar con críticas respecto a la utilidad de los propios estándares, muy especialmente con el perfil MARTE, sobre su tamaño, dificultad de uso… En ese sentido, como una conclusión de esta tesis he de decir que MARTE proporciona al diseñador una variada y versátil capacidad expresiva, rica en elementos, capaz de poder capturar todos los aspectos estructurales y escenarios de análisis para el diseño de sistemas empotrados. A lo largo de esta tesis, se ha mostrado en los artículos publicados y en los manuales desarrollados el papel de MARTE como perfil

de referencia para el modelado de sistemas empotrados ha quedado ampliamente demostrado.

Es labor de trabajos como este la de procesar toda esa cantidad de información y extraer de ella los recursos expresivos mínimos y necesarios para ser aplicados a un área concreta de diseño, creando metodologías de modelado para su uso, restringiendo el número de elementos usados, e implementando herramientas que usen dichos elementos y que implementen las reglas de diseño definidas por esa metodología.

Por tanto, a tenor del trabajo desarrollado durante esta tesis, la gran cantidad de estereotipos, atributos, lenguajes como VSL, que contiene MARTE lo hacen ser un perfil versátil, válido para poder abordar el diseño completo de los sistemas empotrados. Su tamaño no es un punto negativo; es una gran ventaja por las posibilidades expresivas que proporciona. Sin embargo, aún con toda su extensión, presenta algunas pequeñas limitaciones expresivas (por ejemplo, semánticas de comunicación con sus propiedades asociadas) que esta tesis ha complementado.

Una vez en este punto, con el fin de proporcionar y crear entornos de diseño para los ingenieros, esta tesis ha demostrado como se pueden desarrollar herramientas que, basadas en esas metodologías, hagan posible su aplicación de forma fácil. No solo eso, sino también se ha visto cómo es posible que este entorno se conecte con todas las herramientas auxiliares necesarias para realizar los diferentes análisis considerados y permitir la síntesis HW/SW que sea pertinente en cada caso.

En este punto entra otra cuestión a considerar, el cómo de amigables y de fácil manejo sea la metodología desarrollada y las herramientas asociadas para con el usuario. Una crítica que ha ido surgiendo a lo largo del desarrollo de esta tesis es el constante incremento en la complejidad de la metodología como resultado del gran número de etapas a cubrir, haciéndola muchas veces difícil de entender y de aplicar. Sin embargo, esta complejidad es algo difícilmente evitable. Cuando uno pretende abordar el diseño de sistemas empotrados (con todas sus propiedades asociadas al software, hardware y mapeo), su análisis (ej. definición del espacio de diseño a explorar con sus múltiples variantes a explorar) e, incluso, su prototipado (ej. generación de canales de comunicación, estructura concurrente…), toda la información que es necesaria manejar es enorme, lo que implica que el número de recursos expresivos necesarios sea alto. No es lo mismo hacer un modelo donde se describan, más o menos en detalle, una cierta aplicación como forma de documentar, a que se quiera, a partir del modelo, generar automáticamente todo el código a ejecutar en una plataforma. Información como

compiladores, librerías auxiliares, opciones de compilación, código funcional… hace que aumente la carga de los aspectos metodológicos a saber manejar.

Por tanto, dicha complejidad está relacionada con el nivel de detalle que uno desee para su diseño y con el objetivo final del modelo. Para esto se ha de integrar un mayor número de herramientas en el entorno de desarrollo, con el fin de dotar al diseñador de la capacidad de poder realizar la tarea que el considere oportuno. La versatilidad del entorno de desarrollo es, por tanto, uno de los objetivos que ha pretendido cubrir esta tesis.

Así, el conjunto de interfaces de usuario que se han ido desarrollando durante esta tesis siempre ha tenido como objetivo dos premisas: ser fáciles y manejables. En algunas ocasiones, esto no se logró completamente, modificando dicho interfaz en función de las críticas recibidas.

Entrando en más detalle, en la tesis se ha trabajado considerando que durante el desarrollo de este tipo de sistemas se ha de abordar una gran variedad de aspectos, cada uno de ellos con sus propiedades específicas. Este trabajo ha presentado cómo UML y MARTE son lenguajes que lo permiten. Por ejemplo, en lo que se refiere a la funcionalidad, se han de especificar una gran cantidad de información, desde tipos de dato, servicios de interfaz, componentes, funcionalidad asociada a estos, mecanismos y semántica de comunicación...

Además de esto, la concurrencia ya es algo a tener muy en cuenta; forma ya parte del día a día del diseñador y, por tanto, es ya una propiedad del sistema que debe ser considerada como capital. UML y MARTE permiten la posibilidad de definir la estructura concurrente de forma explícita, modelando las tareas presentes en nuestro sistema, e implícita, como consecuencia de la semántica de comunicación de los canales usados para conectar los componentes que conforman la aplicación. A partir de este modelo, se pueden establecer diferentes estrategias para poder estudiar el diseño de nuestro sistema, y ver su rendimiento. Lo relevante de esto es que desde el mismo modelo se puede realizar una simulación funcional, generando una especificación ejecutable que responda a la estructura capturada en el modelo y a las propiedades descritas en él, o establecer un proceso de síntesis de SW a partir del propio modelo. En ambos casos, se puede realizar una ejecución funcional, permitiendo un estudio y análisis. Si esta tarea no es satisfactoria, basta con modificar el modelo y realizar el nuevo análisis.

Además de esto, este estudio funcional del sistema puede ser agnóstico de la plataforma, o considerar dicha plataforma como elemento de nuestro sistema, con el fin de obtener unos resultados más próximos con la implementación final.

En el primer caso, esta simulación se puede ver como una primera refinamiento de la funcionalidad a implementar, sobre todo en busca de indeterminismo, detectando bloqueos u otros problemas y, como anteriormente mencionado, explorando diferentes semánticas de comunicación que definan las propiedades de dicha funcionalidad.

En este contexto, la regla de modelado de separar comunicación y concurrencia es indispensable. Como se mostrado en algunos de los trabajos anteriores, el caso de no hacer está distinción hace del modelado muy restrictivo y nada explorable con respecto a esas propiedades. Esto hace perder una gran cantidad opciones de diseño. En cambio, la separación comunicación-concurrencia permite ampliar el abanico de propiedades a explorar en un sistema, sin obviar la interrelación entre ellas, dotando de más posibilidades de diseño que, de un principio, pudieran no considerarse. Los ejemplos que se han utilizado en los trabajos que sustentan esta tesis han mostrado cuanto de impacto tiene uno u otro tipo de canal, más si cabe aun cuando se dispone de una plataforma multiprocesador.

En este tipo de validación funcional, por sí misma, no puede darse por definitiva en el contexto de los sistemas empotrados. Es adecuado considerar en algún momento la plataforma destino, bien de manera virtual, usando herramientas que proporcionan un entorno de simulación completo (ej. SCoPE) o utilizando la infraestructura necesaria (compiladores, librerías…) para generar el código específico para la plataforma destino.

En este contexto de hacer específicos la plataforma y/o los recursos necesarios para su uso, UML y MARTE tienen una gran capacidad expresiva para poder capturar la plataforma HW/SW y permitir el mapeo de la funcionalidad en ella. El poder expresar procesadores, memorias, buses… y el conjunto de propiedades que los caracterizan hace que la visión de completitud que da el modelo del sistema sea grande y detallada. Es evidente que el nivel de detalle que se quiera expresar depende de los requerimientos del diseñador. Lo que queda claro es que UML y MARTE permiten graduar el nivel de detalle de cada elemento del modelo a las necesidades del usuario y del contexto de diseño.

Por tanto, en esta tesis se ha mostrado lo versátil que es UML/MARTE para poder modelar una gran variedad de sistemas con una gran complejidad y durante diferentes fases en el proceso de diseño, cada una de ellas con sus propiedades específicas; fases como validación funcional, simulación de alto nivel, exploración del espacio de diseño, análisis de planificabilidad, síntesis de SW, síntesis de HW.

Todas ellas son realizadas desde un mismo modelo de UML/MARTE, lo que permite un alto control y una gran trazabilidad del sistema a lo largo del proceso de

desarrollo. Esta trazabilidad permite conservar todos los aspectos del sistema, capturando no solo la definición de las diferentes versiones de los componentes de aplicación o HW/SW que se han usado o se pretende usar, sino también los diferentes tipos de análisis y escenarios de los mismos que se realizarán. Además de esto, el modelo UML/MARTE permite mantener un historial de la evolución del diseño, los cambios hechos, las optimizaciones realizadas y también los errores cometidos. Todo esto hace más fácil la evolución tecnológica de un producto, y una adaptación a dichos cambios más fácil y eficiente. Todo esto también se ve favorecido por la versatilidad de UML y de MARTE ya que, debido a su riqueza expresiva, permiten adaptar metodologías ya asentadas con nuevos mecanismos expresivos para capturar nuevos requerimientos del sistema.

Finamente, el hecho de partir desde un modelo UML/MARTE facilita la colaboración entre las personas encargadas del diseño, cuando este se realiza en grupo o mediante la colaboración de varios grupos de trabajo. A veces, el hecho de que cada agente implicado en el diseño del sistema se enfoque en su área específica de desarrollo hace que no se tenga en cuenta otras con gran impacto en el resultado, como se ha mostrado en los ejemplos mostrados en esta tesis. El modelo UML/MARTE aporta esta visión global del sistema, permitiendo estudiar el sistema de forma compartimentada, cada área por separado, y en conjunto, tratando el sistema como un todo. De esta forma, el diseño se puede organizar en sub-tareas pero sin perder de vista la completitud del sistema.

De la misma manera, el modelo UML/MARTE permite la reutilización del diseño, pudiendo exportar e importar elementos de un modelo a otro, creando librerías de elementos que pueden ser usados en otros diseños.

Hay que añadir que, con el fin de intentar ser siempre lo más didáctico posible para explicar todos los aspectos de la metodología, he tratado siempre de que en los artículos y en los manuales realizados durante el desarrollo de esta tesis fuesen lo más claro y sencillo de entender posible para el potencial usuario.

Por último, he de incidir en lo colaborativo del trabajo presentado en esta tesis, que se ha visto sustentado por la acción de diferentes personas que han aportado mucho en la realización del trabajo presentado en esta tesis.

Capítulo VIII

Referencias: Artículos Científicos en los que he Colaborado

Capítulo VIII. Referencias: Artículos Científicos en los he Colaborado

En esta sección se listan todas los artículos que se han publicado a lo largo del desarrollo de esta tesis. Algunos de ellos han sido utilizados como referencia a lo largo de este documento para denotar donde el lector puede encontrar más detalles del tema tratado.

Esta sección se organiza de la siguiente forma:

1. Artículos de Revistas.
2. Capítulos de Libro.
3. Artículos de Conferencias.
4. Libro publicado.

Además de esto, esta sección incluye las referencias al conjunto de manuales que se han ido desarrollando durante la realización de esta tesis, que han sido usados como documentación auxiliar a lo presentado en este documento.

En la última sección, se mostrarán los proyectos de investigación en el desarrollo de cuales, el trabajo de esta tesis ha sido elaborado.

1. Revistas

a) High-Level design of wireless sensor networks form performance optimization under security hazards. P. Peñil, Á. Díaz, H. Posadas, J. Medina, P. Sánchez. Transactions on Sensor Networks (TOSN) ACM. ACEPTADA.

b) The COMPLEX methodology for UML/MARTE modeling and design-space exploration of embedded systems. F. Herrera, H. Posadas, P, Peñil, E. Villar, F. Ferrero, R. Valencia, G. Palermo. Journal of Systems Architecture, V.60, N.1, Elsevier, pp.55–78. 2014-01.

c) Automatic synthesis of embedded SW for evaluating physical implementation alternatives from UML/MARTE models supporting memory space separation. H. Posadas, P. Peñil, A. Nicolás, E. Villar. Microelectronics Journal. Volume 45, Issue 10, Octubre 2014, páginas 1281–1291.

d) Synthesis of Simulation and Implementation Code for OPENMAX Multimedia Heterogeneous Systems from UML/MARTE models. D. de la Fuente, P. Peñil, J. Barba, H. Posadas, J.C. López, P. Sánchez. Multimedia Tools & Applications. pp 1–32. Abril 2016.

e) Generating Heterogeneous Executable Specifications in SystemC from UML/MARTE Models. P. Peñil, J. Media, H. Posadas, E. Villar. "Innovations in Systems and Software Engineering". Marzo 2010, Volume 6, Issue 1, pp 65–71.

f) Automatic synthesis of communication and concurrency for exploring component-based system implementations considering UML channel semantics. H. Posadas, P. Peñil, A. Nicolás, E. Villar. Journal of System Architecture, V.61, I.8, pp.341–360. 2015-09.

g) Improving the Design Flow for Parallel and Heterogeneous Architectures running Real-Time applications: The PHARAON FP7 project. H. Posadas, A. Nicolás, P. Peñil, E. Villar, F. Broekaert, M. Bourdelles, A. Cohen, M. T. Lazarescu, L. Lavagno, A. Terechko, M. Glassee, M. Prieto. Microprocessors and Microsystems, V.38, I.8, Part B, pp. 960–975. 2014-11.

h) CONTREX: Design of embedded mixed-criticality CONTRol systems under consideration of EXtra-functional properties. K. Grüttner, R. Görgen, S. Schreiner, F. Herrera, P. Peñil, J. Medina, E. Villar, G. Palermo, W. Fornaciari, C. Brandolese, D. Gadioli, E. Vitali, D. Zoni, S. Bocchio, L. Ceva, P. Azzoni, M. Poncino, S. Vinco, E. Macii, S. Cusenza, J. Favaro, R. Valencia, I. Sander, K. Rosvall, N. Khalilzad, D. Quaglia. Microprocessors and Microsystems, Abril 2017. http://doi.org/10.1016/j.micpro.2017.03.012.

2. Capítulo de Libro

i) Formal Foundations for the Generation of Heterogeneous Executable Specifications in SystemC from UML/MARTE Models. P. Peñil, F. Herrera, E. Villar. Embedded systems Theory and Design Methodology. ISBN 978-953-51-0167-3. InTech, Croacia. 2012-02.

j) Formal Support for Untimed MARTE-SystemC Interoperability. P. Peñil, F. Herrera, E. Villar. T. Kazmierski & A. Morawiec (Eds.): "Systems Specification and Design Languages", Lecture Notes in Electrical Engineering, V.106, Springer. 2012-06.

k) Model-Driven Methodology for the Development of Multi-level Executable Environments. F. Herrera, P. Peñil, H. Posadas, E. Villar. J. Haase (Ed.): "Models, Methods and Tools for Complex Chip Design", Lecture Notes in Electrical Engineering, V.265, Springer. 2014-01.

l) The SATURN Approach to SysML-Based HW/SW Codesign. W. Mueller, D. He, F. Mischkalla, A. Wegele, A. Larkham, P. Whiston, P. Peñil, E. Villar, N. Mitas, D. Kritharidis, F. Azcarate, M. Carballeda. N. Voros, A. Mukherjee, N. Sklavos, K. Masselos, M. Huebner (Eds.): "VLSI 2010 Annual Symposium Selected Papers", Lecture Notes in Electrical Engineering, V.57, pp. 151-164, Springer. 2011-10

3. Artículos de Conferencias

m) Formal Modeling for UML/MARTE Concurrency Resources. Pablo Peñil, Héctor Posadas, Eugenio Villar. Proc. of the 15th IEEE International Conference on Engineering of Complex Computer Systems. 24/03/2010. ISBN 9780769540153.

n) Formal Foundations for MARTE-SystemC Interoperability. P. Peñil, F. Herrera, E. Villar. Proceedings of the Forum on specification & Design Languages (FDL 2010). FDL, page 197-202. ECSI, Electronic Chips & Systems design Initiative, (2010).

o) Enhancing Analyzability and Time Predictability in UML/MARTE Component-based Application Models. F. Herrera, P. Peñil, E. Villar. Forum on specification & Design Languages, FDL 2015. 15/09/2015.

p) UML/MARTE methodology for high-level system estimation and optimal synthesis. H. Posadas, P. Peñil, A. Nicolás, E. Villar. MeCoES - Metamodeling and Code Generation for Embedded Systems, ESWeek 2012. 2012-10.

q) System level design framework for many-core architectures. 3 PMCES 2014. P. Peñil, L. Díaz, P. Sánchez. 28/03/2014.

r) Modeling and Simulation of secure wireless sensor network. Forum on specification & Design Languages, FDL 2012. P. Peñil, L. Díaz, P. Sánchez, J. Sancho, J. Rico. Fecha: 20/09/2012. ISBN 978-1-4673-1240-0.

s) UML-Based Single-Source Approach for Evaluation and optimization of Mixed-Critical Embedded Systems. P. Peñil, H. Posadas, J. Medina, E. Villar. XXX Conference on Design of Circuits and Integrated Systems, DCIS 2015, IEEE. 2015-11.

t) Building a Dynamically Reconfigurable System through a High-Level Development Flow. D. de la Fuente, J. Barba, J. C. López, X. Peña, P. Peñil and P. Sánchez. Forum on specification & Design Languages, FDL 2015. 15/09/2015.

u) UML/MARTE Modelling for Design Space Exploration of Mixed-Criticality Systems on top of Time-Predictable HW/SW Platforms. F. Herrera, P. Peñil, E. Villar. Jornadas de Computación Empotrada (JCE15). 2015-09.

v) Automatic Synthesis over multiple APIs from UML/MARTE Models for easy Platform Mapping and Reuse. A. Nicolás, P. Peñil, H. Posadas, E. Villar. Proceedings of the EuroMicro DSD/SEAA Conference, IEEE, 2014. 2014-08.

w) A model-based, single-source approach to design-space exploration and synthesis of mixed-criticality systems. Fernando Herrera, P. Peñil, E. Villar. 18th International Workshop on Software and Compilers for Embedded Systems, SCoPES 2015, ACM. 2015.

x) Automatic Deployment Of Component-Based Embedded Systems From UML/MARTE Models Using MCAPI, A. Nicolás, P. Peñil, H. Posadas, E. Villar. XXIX Conference on Design of Circuits and Integrated Systems, DCIS 2014. 2014-11.

y) Automatic Concurrency generation through Communication Data Splitting based on UML-MARTE Models. A. Nicolás, H. Posadas, P. Peñil, E. Villar. XXVIII Conference on Design of Circuits and Integrated Systems, San Sebastian, Noviembre, 2013.

z) Easy Modeling and Fast Exploration of Multimedia Heterogeneous Applications with UML/MARTE. David de La Fuente, Jesús Barba, P. Peñil, P. Sánchez, Julio Daniel Dondo, María José Santofimia. XXVIII Conference on Design of Circuits and Integrated Systems, San Sebastian, Noviembre, 2013.

aa) Code Synthesis of UML/MARTE models for physical platforms considering resource-specific codes. P. Peñil, H. Posadas, A. Nicolás, E. Villar, D. Calvo. IV Jornadas de Computación Empotrada, Sarteco 2013. 2013-09.

bb) UML/MARTE methodology for automatic SystemC code generation of OPENMAX multimedia applications. P. Peñil, P. Sánchez, D. de la Fuente, J. Barba, J.C. López. Euromicro Conference on Digital System Design, DSD 2013, IEEE. 2013-09.

cc) System synthesis from UML/MARTE models: The PHARAON approach. H. Posadas, P. Peñil, A. Nicolás, E. Villar. Electronic System Level Synthesis Conference, ESLsyn, 2013, IEEE. 2013-05.

dd) Automatic synthesis of Embedded SW Communications from UML/MARTE models supporting memory-space separation. H. Posadas, P. Peñil, A. Nicolás, E. Villar. XXVII Conference on Design of Circuits and Integrated Systems, DCIS'12. 2012-11.

ee) A MDD Methodology for Specification of Embedded Systems and Automatic Generation of Fast Configurable and Executable Performance Models. F. Herrera, H. Posadas, P. Peñil, E. Villar, F. Ferrero, R. Valencia. ESWeek 2012 Compilation Proceedings, CoDes+ISSS'12, ACM. 2012-10.

ff) The Complex Eclipse Framework for UML/MATE Specification and design Space Exploration of Embedded Systems. F. Herrera, H. Posadas, P. Peñil, E. Villar, F. Ferrero, R. Valencia. Proceedings of the 2012 Conference on Design & Architectures for Signal & Image Processing, DASIP 2012, IEEE. 2012-10.

gg) An Embedded System Modelling Methodology for Design Space Exploration. F. Herrera, P. Peñil, E. Villar, F. Ferrero, R. Valencia. III Jornadas de Computación Empotrada, Sarteco 2012. 2012-09.

hh) A Model-Driven Methodology for the Development of SystemC Executable Environments. F. Herrera, P. Peñil, H. Posadas, E. Villar. Proceedings of the 2012 Forum on Specification and Design Languages, FDL'2012, IEEE. 2012-09.

ii) Automatic synthesis from UML/MARTE models using channel semantics. P. Peñil, H. Posadas, A. Nicolás, E. Villar. International Workshop on Model-Based Arquitecting and Construction of Embedded Systems, ACES-MB 2012-09.

jj) Towards SystemC Code Generation from UML/MARTE Concurrent System-Level Models. P. Peñil, F. Herrera, E. Villar. W6: 2nd Workshop on Model Based Engineering for Embedded Systems Design, DATE 2011. 2011-03.

kk) Executable SystemC specification of the MARTE generic concurrent and communication resources under different Models of Computation. P. Peñil, E. Villar, H. Posadas, J. Medina. Workshop on the Definition, evaluation, and exploitation of modelling and computing standards for Real-Time Embedded Systems, STANDRTS'09 Satellite Workshop of the the 21st EuroMicro Conference on Real-Time Systems, Dublin. 2009-06.

ll) R. Görgen, K. Grüttner, F. Herrera, P. Peñil, J.L. Medina, E. Villar, G. Palermo, W. Fornaciari, C. Brandolese, D. Gadioli, S. Bocchio, L. Ceva, P. Azzoni, M. Poncino, S. Vinco, E. Macii, S. Cusenza, J. Favaro, R. Valencia, I. Sander, K. Rosvall, D. Quaglia. CONTREX: design of embedded mixed-criticality control systems under consideration of extra-functional properties. P. Kitsos (Ed.), 2016 Euromicro Conference on Digital System Design, DSD 2016, Limassol, Cyprus, August 31 - September 2, 2016, IEEE Computer Society (2016), pp. 286–293 http://doi.org/10.1109/DSD.2016.95

4. Libro

mm) Generación de Especificaciones Ejecutables SystemC Heterogéneas. Pablo Peñil. EAE Editorial Academia Española. 4 de julio de 2012. ISBN-10: 3848469774. https://www.amazon.es/Generacion-Especificaciones-Ejecutables-Systemc-Heterogeneas/dp/3848469774/ref=sr_1_1?ie=UTF8&qid=1490172209&sr=8-1&keywords=pablo+pe%C3%B1il

5. Documentación Adicional: manuales

nn) Introducción a la Metodología de Modelado UML/MARTE. http://umlmarte.teisa.unican.es/wp-content/uploads/2016/05/UML-MARTE_Methodology_introduction.pdf

oo) Metodología de Modelado UML/MARTE. http://umlmarte.teisa.unican.es/wp-content/uploads/2016/09/UML-MARTE_Methodology_for_Embedded_Systems_modelling.pdf

pp) Metodología UML/MARTE para sistemas distribuidos. http://umlmarte.teisa.unican.es/wp-content/uploads/2016/05/ExtendedUML_MARTE_Network_Modelling_Methodology.pdf

qq) Metodología UML/MARTE para DSE. http://umlmarte.teisa.unican.es/wp-content/uploads/2016/05/UML-MARTE_Methodology_for_dse.pdf

rr) Metodología UML/MARTE para Síntesis. http://umlmarte.teisa.unican.es/wp-content/uploads/2016/05/UML-MARTE_Methodology_for_synthesis.pdf

ss) Metodología UML/MARTE para Análisis de Planificabilidad. http://umlmarte.teisa.unican.es/wp-content/uploads/2016/05/UML-MARTE_Methodology_for_schedulability.pdf

tt) Metodología UML/MARTE para Sistemas de Criticidad Mixta. http://umlmarte.teisa.unican.es/wp-content/uploads/2016/05/UML_MARTE_Mixed_Criticalities.pdf

uu) Metodología para la interoperabilidad con modelos SDF. http://umlmarte.teisa.unican.es/wp-content/uploads/2016/05/UML_MARTE_ForSyDe_SDF_2016_05_31.pdf

6. *Proyectos*

Esta tesis se ha desarrollo en el contexto de un conjunto de proyectos de investigación internacionales y nacionales. En cada una de ellos se ha desarrollado una parte del entorno presentado en esta tesis, añadiendo nuevos aspectos a partes ya realizadas. Esta información ha sido detalla en cada sección del documento.

vv) Proyecto SATURN (SysML bAsed modeling, architecTUre exploRation, simulation and syNthesis for complex embedded systems): https://www.teisa.unican.es/gim/en/proyecto?id=69

ww) Proyecto CONTREX (Design of embedded mixed-criticality CONTRol systems under consideration of EXtra-functional properties): https://contrex.offis.de/home/. http://bree.teisa.unican.es/gim/proyecto?id=101

xx) Proyecto COMPLEX (COdesign and Power Management in PLatform-Based Design Space EXploration): https://www.teisa.unican.es/gim/es/proyecto?id=87

yy) Proyecto TOISE (Trusted Computing for European Embedded Systems): http://gim.teisa.unican.es/es/proyecto?id=90

zz) Proyecto DREAMS (Dynamically Reconfigurable Embedded Platforms for Networked Context-Aware Multimedia Systems): http://gim.teisa.unican.es/es/proyecto?id=96

aaa) Proyecto PHARAON (Parallel and Heterogeneous Architecture for Real-time ApplicatiONs): https://www.teisa.unican.es/gim/en/proyecto?id=95

bbb) Proyecto CRAFTERS (Constraint and Application driven Framework for Tailoring Embedded Real-time Systems): https://www.teisa.unican.es/gim/es/proyecto?id=98

Capítulo IX

Referencias

[1] M. Panunzio, T. Vardanega. On Component-based development and high integrity real-time systems, in: Proceedings of the 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, ERTCS-09, Beijing, China, 2009.

[2] “UML Specification 2.5”. OMG, ww.omg.org/spec/UML/, June, 2015.

[3] "UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems Version 1.1", OMG, www.omgmarte.org, June, 2011.

[4] OMG: "MDA guide, Version 1.1", June 2003.

[5] J. L. Medina, A. Pérez. High Level Modeling for Real-Time Applications with UML & MARTE 25th Euromicro Conference on Real-Time Systems (ECRTS’13) WiP Session, pp. 13-16, July 2013.

[6] J. Medina, A. García Cuesta. Model-Based Analysis and Design of Real-Time Distributed Systems with Ada and the UML Profile for MARTE. 6th Int. Conf. On Reliable Software Technologies, Ada-Europe'2011, Edinburg (UK), in Lecture Notes in Computer Science, LNCS Vol. 6652, pp. 89-102, June 2011.

[7] J. L. Medina, A. Pérez. High Level Modeling for Real-Time Applications with UML & MARTE 25th Euromicro Conference on Real-Time Systems (ECRTS’13) WiP Session, pp. 13-16, July 2013.

[8] E. Lee: "The problem with threads", Computer, May, 2006.

[9] A. Jantsch. "Modeling Embedded Systems and SoCs", Morgan Kaufmann, 2004.

[10] F.Herrera, E. Villar. "A Framework for Heterogeneous Specification and Design of Electronic Embedded Systems in SystemC" ACM Transactions on Design Automation of Electronic Systems, Special Issue on Demonstrable Software Systems and Hardware Platforms, V.12, Issue 3, N.22. 2007-08.

[11] SystemC-ForSyDe. https://forsyde.ict.kth.se/trac/wiki/ForSyDe/SystemC.

[12] Artisan Studio. http://www.atego.com/products/sysim/

[13] Papyrus. https://projects.eclipse.org/projects/modeling.mdt.papyrus

[14] Acceleo. http://www.eclipse.org/acceleo/.

[15] MOFM2t. http://www.omg.org/spec/MOFM2T/1.0/

[16] Eclipse. https://eclipse.org/

[17] OpenMAX. https://www.khronos.org/openmax/.

[18] Grupo ARCO, UCLM. http://arco.esi.uclm.es/

[19] SCoPE website. www.teisa.unican.es/scope. Dec., 2012.

[20] J. Castillo, H. Posadas, E. Villar, M. Martínez. "Fast Instruction Cache Modeling for Approximate Timed HW/SW Co-Simulation". 20th Great Lakes Symposium on VLSI (GLSVLSI'10), Providence, USA. 2010-05.

[21] L. Diaz, E. González, E. Villar, P. Sánchez "VIPPE: Parallel simulation and performance analysis of complex embedded systems" HiPPES4CogApp: High Performance, Predictable Embedded Systems for Cognitive Application. 2015-01.

[22] Acceleo. http://www.eclipse.org/acceleo/.

[23] Á. Díaz, P. Sánchez, J. Sancho, J. Rico "Wireless Sensor Network Simulation for Security and Performance Analysis" DATE 2013. 2013-03.

[24] MAST, www.mast.unican.es.

[25] Google Tests, http://www.ibm.com/developerworks/aix/library/au-googletestingframework.html

[26] eSSYN, www.essyn.com/

[27] MOST, http://conferenze.dei.polimi.it/depcp/2011/proceedings/pdfs/18.pdf

[28] SystemC, http://accellera.org/downloads/standards/systemc

[29] Xilinx, https://www.xilinx.com/

[30] OMG. UML Testing Profile (UTP) 1.1 website http://utp.omg.org/ . Nov 2014.

[31] Ralph Gorgen, Kim Grutner, F. Herrera et al. CONTREX: Design of embedded mixed-criticality CONTRol systems under consideration of EXtra-functional properties. Euromicro DSD 2016. Limassol, Cyprus. 2016-08

[32] K. Grüttner, P.A. Hartmann, K. Hylla, S. Rosinger, W. Nebel, F. Herrera, E. Villar, C. Brandolese, W. Fornaciari, G. Palermo, C. Ykman-Couvreur, D. Quaglia, F. Ferrero (GMV), R. Valencia (GMV). The COMPLEX reference framework for HW/SW co-design and power management supporting platform-based design-space exploration. Microprocessors and Microsystems, V.37, N.8-C, Elsevier, pp.966-80. 2013-11.

[33] E.Villar, F.Herrera and V. Fernández. Formal support for Untimed SystemC Specifications: Application to high-level synthesis. In Proceedings of FDL'2010. September, 2010.

[34] UML for real: design of embedded real-time systems. L. Lavagno, G. Martin, B. Selic. ISBN 1-4020-7501-4.

[35] Y. Vanderperren, W. Mueller, W. Dehaene. UML for Electronic Systems Design: a comprehensive overview. Journal on Design Automation for Embedded Systems, Springer Verlag, August 2008.

[36] J. Vidal, F. de Lamotte, G. Gogniat, P. Soulard, J.P. Diguet. A code-design approach for embedded system modeling and code generation with UML and MARTE. Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE'09.

[37] É. Piel, R. Atitallah, P. Marquet, S. Meftali, S. Niar, A. Etien, J.-L. Dekeyser, P. Boulet, ''Gaspard2: from MARTE to SystemC Simulation'', Proc. of the DATE'08 workshop, 2008.

[38] J.L.Dekeyser, A. Gamatié, A. Etien, R.B.Atitallah, P. Boulet. "Using the UML Profile for MARTE to MPSoC Co-Design". In First International Conference on Embedded Systems & Critical Applications (ICESCA'08), Tunis, Tunisia, May 2008.

[39] I.R. Quadri, Yu Huafeng, A. Gamatie, E. Rutten, S. Meftali, J.-L. Dekeyser. Targeting reconfigurable FPGA based SoCs using the UML MARTE profile: from high abstraction levels to code generation. International Journal of Embedded Systems, v 4, n 3-4, p 204-24, 2010.

[40] S. Taha, A. Radermacher, S. Gerard, J.L. Dekeyser. MARTE: UML-based Hardware Design from Modeling to Simulation. FDL'2007, Barcelona, Spain, September 2007.

[41] C. Mraidha, Y. Tanguy, C. Jouvray, F. Terrier, S. Gérard. An Execution Framework for MARTE-based Models. Thirteenth IEEE International Conference on the Engineering of Complex Computer Systems, ICECCS 2008.

[42] A. Koudri, J. Champeau, J.-C. Le Lann, V. Leilde. MopCom Methodology: Focus on Models of Computation. Modelling Foundations and Applications. Proceedings 6th European Conference, ECMFA 2010, p 189-200, 2010.

[43] E. Riccobene, P. Scandurra, A. Rosti and S. Bocchio. A UML 2.0 Profile for SystemC. STMicroelectronics Technical Report, 2004.

[44] R. Eshuis, R.Wieringa. A Formal Semantics for UML Activity Diagrams– Formalising Workflow Models. CTIT technical reports series (01-04). ISSN 13813625.

[45] H. Störrle, J.H. Hausmann. Towards a Formal Semantics of UML 2.0 Activities. In Proc. Software Engineering 2005, 2005.

[46] T. Bouabana-Tebibel, M. Belmesk. An object-oriented approach to formally analyze the UML 2.0 activity partitions. Information and Software Technology 49 (9-10), pp. 999-1016.

[47] R. Eshuis, R.Wieringa. A Formal Semantics for UML Activity Diagrams– Formalising Workflow Models. CTIT technical reports series (01-04). ISSN 13813625.

[48] C. Eichner, H. Fleischhack, R. Meyer, U. Schrimpf, C. Stehno. Compositional Semantics for UML 2.0 Sequence Diagrams Using Petri Nets. Lecture Notes in Computer Science, v 3530, p 133-148, 2005, SDL 2005.

[49] J. K. F. Bowles and B. Bordbar. A Formal Model for Integrating Multiple Views. Application of Concurrency to System Design, 2007. ACSD 10-13 July 2007 Page(s):71 – 79.

[50] F. Mallet, C. André. On the semantics of UML/MARTE Clock Constraints. International Symposium on Object/component/ service/oriented. Real-time distributed Computing (ISORC) (2009) 301-312.

[51] F. Mallet. Clock constraint specification language: specifying clock constraints with UML/MARTE. Innovations in Systems and Software Engineering, Vol. 4, No. 3. (1 October 2008), pp. 309-314.

[52] C. André, F. Mallet, R. de Simone. Modeling Time(s). In G. Engels, B. Opdyke, D. C. Schmidt, and F. Weil, editors, MoDELS, volume 4735 of Lecture Notes in Computer Science, pages 559–573. Springer, 2007.

[53] C. André, F. Mallet, R. de Simone. Modeling of immediate vs. delayed data communications: from AADL to UML MARTE. ECSI FDL 2007.

[54] Mueller, W., Ruf, J., Hoffmann, D., Gerlach, J., Kropf, T., Rosenstiehl, W.: The Simulation Semantics of SystemC. In Proc. of DATE'01. IEEE. (2001).

[55] Kroening, D. and Sharygna, N.: Formal Verification of SystemC by Automatic Hardware/Software Partitioning. In Proc. of MEMOCODES'05. Verona, Italy (2005).

[56] Maraninchi, F., Moy, M., Maillet-Contoz, L.: Lussy: An Open Tool for the Analysis of Systems-on-a-Chip at the Transaction Level. In Design Automation of Embedded Systems, 10(2-3):pp.74–103, September (2005).

[57] Traulsem, C., Cornet, J., Moy, M., Maraninchi, F.: A SystemC/TLM semantics in PROMELA and its Possible Applications. In 14th Workshop on Model Checking Software (SPIN'07), July (2007).

[58] Mueller, W., Ruf, J., Hoffmann, D., Gerlach, J., Kropf, T., Rosenstiehl, W.: The Simulation Semantics of SystemC. In Proc. of DATE'01. IEEE. (2001).

[59] Kroening, D. and Sharygna, N.: Formal Verification of SystemC by Automatic Hardware/Software Partitioning. In Proc. of MEMOCODES'05. Verona, Italy (2005).

[60] Maraninchi, F., Moy, M., Maillet-Contoz, L.: Lussy: An Open Tool for the Analysis of Systems-on-a-Chip at the Transaction Level. In Design Automation of Embedded Systems, 10(2-3):pp.74–103, September (2005).

[61] Traulsem, C., Cornet, J., Moy, M., Maraninchi, F.: A SystemC/TLM semantics in PROMELA and its Possible Applications. In 14th Workshop on Model Checking Software (SPIN'07), July (2007).

[62] D. Harel, H. Kugler, A. Pnueli, Synthesis revisited: Generating statechart models from scenario-based requirements, Formal Methods Softw. Syst. Model. (2005).

[63] M. Adamski, Design of reconfigurable logic controllers from hierarchical UML state machines, in: Proceedings of the 2009 4th IEEE Conference on Industrial Electronics and Applications, ICIEA 2009.

[64] S. Kang, H. Kim, J. Baik, H. Choi, C. Keum, Transformation Rules for Synthesis of UML Activity Diagram from Scenario-Based Specification, in: IEEE Proceedings of 34th Annual Computer Software and Applications Conference (COMPSAC), 2010.

[65] J. Barba, F. Rincón, F. Moya, D. Villa, F.J. Villanueva, J.C. López. "Automatic HW/SW Interface Generation for Seamless Integration from Object-Oriented Models", in: Proceedings of the International Conference on Embedded Systems and Applications. ESA, 2009.

[66] <http://msdn.microsoft.com/en-us/library/vstudio/ff657795.aspx>.

[67] <http://www.ibm.com/developerworks/rational/library/08/0826_makady/>.

[68] M. Leite, C.D. Vasconcellos, M.A. Wehrmeister (2014) Enhancing automatic generation of VHDL descriptions from UML/MARTE models. 12th IEEE International Conference on Industrial Informatics (INDIN) Lukas S Staff Engineer, QuIC, Inc. Accessing hardware-accelerated video codecs on Android. UPLINQ.

[69] I.R. Quadri, H. Yu, A. Gamatié, E. Rutten, S. Meftali, J.L. Dekeyser. Targeting reconfigurable FPGA based SoCs using the UML MARTE profile: from high abstraction levels to code generation. International Journal of Embedded Systems, Inderscience, 2010, 18 p.

[70] S. Zhenxin, W. Weng-Fai, ''A UML-based approach for heterogeneous IP integration'', ASP-DAC, 2009.

[71] C.-S. Peng, L.-C. Chang, C.-H. Kuo, B.-D. Liu "Dual-core virtual platform with QEMU and SystemC". 2010 International Symposium on Next-Generation Electronics, ISNE 2010-Conference Program. OMG: "UML profile for system on chip (SoC) specification", 2006.

[72] T. Robert, V. Perrier, COFLUENT Methodology for UML: UML SysML MARTE Flow for CoFluent Studio, White paper, 2012. Available from: <http://www.cofluentdesign.com/index.php/solutions/uml-sysml-marte>.

[73] A.W. Liehr, H.S. Rolfs, K.J. Buchenrieder, U. Nageldinger, Generating MARTE allocation models from activity threads, in: Languages for Embedded Systems and their Applications, Lecture Notes in Electrical Engineering, vol. 39, 2009, pp. 43–56 (Part I).

[74] M. Mura, L.G. Murillo, M. Prevostini, Model-based design space exploration for RTES with SysML and MARTE, in: Proceedings of the Forum on Specification and Design Languages 2008, FDL'2008. Stuttgart, Germany, 2008.

[75] T. Kangas et al., UML-based multiprocessor SoC design framework, Journal ACM Transactions on Embedded Computing Systems (TECS) 5 (2) (May 2006) 281–320.

[76] G. Saggio, P. Cavallo, L. Bianchi, L. R. Quitadamo, F. Giannini. UML Model Applied as a Useful Tool for Wireless Body Area Networks. Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology, 2009.

[77] L. Berardinelli, V. Cortellessa, S. Pace Modeling and Analyzing Performance of Software for Wireless Sensor Networks. SESENA 11Proceedings of the 2nd Workshop on Software Engineering for Sensor Network Applications.

[78] E. Ebeid, D. Quaglia, F. Fummi. Generation of SystemC/TLM code from UML/MARTE sequence diagrams for verification. Design and Diagnostics of Electronic Circuits & Systems (DDECS), 2012 IEEE 15th International Symposium.

[79] S. Hong, S. Limt Analysis of Attack Models via Unified Modeling Language in Wireless Sensor Networks: A Survey Study Wireless Communications, Networking and Information Security (WCNIS), 2010 IEEE International Conference.

[80] E. Ebeid, F. Fummi, D. Quaglia. 2013, "UML-based modeling and simulation of environmental effects in networked embedded systems", Proceedings - 16th Euromicro Conference on Digital System Design, DSD 2013, pp. 787.

[81] E. Ebeid, F. Fummi, D. Quaglia, H. Posadas, E. Villar. 2014, "A framework for design space exploration and performance analysis of networked embedded systems", ACM International Conference Proceeding Series.

[82] J. Jürjens. UMLsec: Extending UML for secure Systems Development. UML" 2002 - Unified Modeling Language. Model Engineering, Concepts, and Tools. 5th International Conference. Proceedings (Lecture Notes in Computer Science Vol.2460), p 412-25, 2002.

[83] J. Jurjens, J. Fox. Tools for model-based security engineering.Source: 28th International Conference on Software Engineering Proceedings, p 819-22, 2006.

[84] J. Jürjens. Model-based Security Analysis for Mobile Communications. 2008 ACM/IEEE 30th International Conference on Software Engineering. ICSE'08, p 683-92, 2008.

[85] J. Jürjens. Automated security hardening for evolving UML models. 2011 33rd International Conference on Software Engineering (ICSE 2011), p 986-8, 2011.

[86] D. C. Petriu, C. Murray Woodside, D. B. Petriu, J. Xu, T. Israr, G. Georg, R. B. France, J. M. Bieman, S. H. Houmb, J. Jürjens. Performance Analysis of Security Aspects in UML Models. Journal of Systems and Software, v 82, n 1, p 56-74, Jan. 2009.

[87] A. Diaz, P. Sanchez. Simulation of Attacks for Security in Wireless Sensor Network. Sensors 2016, 16(11), 1932; doi:10.3390/s16111932.

[88] HetSC. http://www.teisa.unican.es/HetSC/

[89] E. Ebeid, F. Fummi, D. Quaglia, and F. Stefanni, “Refinement of UML/MARTE Models for the Design of Networked Embedded Systems,” in Design, Automation Test in Europe Conference Exhibition (DATE), 2012, Mar. 2012, pp. 1072–1077.

[90] E. Ebeid, D. Quaglia, and F. Fummi. “UML-based Modeling and Simulation of Environmental Effects in Networked Embedded Systems”. In 16th Euromicro Conference on Digital System Design (DSD), 2013.

Capítulo IX. Referencias