Sergio Serna

Especificación formal de requisitos temporales no funcionales

Tesis para optar el título de Magister en Ingeniería de Sistemas

Director: Fernando Arango Isaza, PhD.

5 de julio de 2011

Facultad de Minas
Universidad Nacional de Colombia

Ana María, eres mi alegría y mi fortaleza.

Reconocimientos

Este trabajo se realizó con el decidido apoyo de mi tutor Fernando Arango Isaza, PhD., quien lo dirigió de manera muy acertada, y me dio claridad en muchos aspectos.
Gracias al Instituto Tecnológico Metropolitano – ITM, por su permanente apoyo durante estos años en los que he trabajado en esa grandiosa Institución.
Gracias a los demás.

Índice general

1. Introducción 5
1.1. Trabajos derivados de esta tesis 8

2. Marco Teórico 11
2.1. Ingeniería de requisitos 11
2.1.1. Calidad de los requisitos 12
2.1.2. Actividades de la ingeniería de requisitos 13
2.2. Requisitos de software 15
2.2.1. Requisitos funcionales 16
2.2.2. Requisitos no funcionales 17
2.3. UN-METODO para la educción de requisitos de software 20
2.3.1. Generalidades de UN-METODO 20
2.4. El lenguaje de modelado unificado 22
2.4.1. El metamodelo de UML 22
2.4.2. Diagramas UML 24
2.5. Perfiles UML 33
2.5.1. Perfil para modelado y análisis de sistemas embebidos de tiempo real 33
2.6. El lenguaje de especificación de valores 51
2.6.1. Descripción general 51
2.6.2. Perfil UML para VSL 51
2.7. El lenguaje de restricción de objetos 53

3. Antecedentes 55
3.1. Caso de estudio: sistema de vigilancia para un campamento militar 56
3.2. Diferentes soluciones al sistema de vigilancia para un campamento militar 56
3.2.1. Educción de requisitos del sistema de vigilancia para un campamento militar . . 57
3.2.2. Solución: marco de trabajo basado en NFRs 57
3.2.3. Soluciones basadas en UML 64
3.2.4. Solución: NFRs con ontologias 87
3.2.5. Solución: Aspectos 89
3.2.6. Solución: XML 93

3.3. Problemas al tratar con requisitos no funcionales 95
3.3.1. Problemas de definición 95
3.4. Problemas de clasificación 97
3.5. Problemas de representación 98

4. Planteamiento del Problema de Investigación 99
4.1. Justificación 100
4.2. Preguntas de investigación 100
4.3. Objetivos 101
4.3.1. Objetivo general 101
4.3.2. Objetivos específicos 101

5. Método Propuesto 103
5.1. Metodología para la educción de requistos 103
5.1.1. Método para la representación y especificación de NFRs temporales 104
5.2. Especificación formal de NFRs en el diagrama de clases 105
5.2.1. Propuesta semántica 106
5.2.2. Propuesta sintáctica 109

6. Resultados y discusión 115
6.1. Caso de estudio: sistema de vigilancia para un campamento militar 115
6.1.1. Implementación obtenida con el método propuesto en esta tesis 117
6.2. Caso de estudio: sistema de monitoreo Web del FP de un motor DC 117
6.3. Análisis de resultados 122

7. Conclusiones y trabajos futuros 125
7.1. Conclusiones 125
7.2. Trabajos futuros 127

A. Socialización de resultados 129

Glosario 135

Referencias 137

Índice de figuras

2.1. Tipos de requisitos no funcionales, destacando al tiempo como subtipo del desempeño 19
2.2. Diagrama de las actividades desarrolladas en el marco de UN-METODO (adaptado de [Arango:2006]) 21
2.3. Metaniveles diferentes de UML y MOF [UMLInfra:2009] 23
2.4. Metamodelo de 4 capas para UML y MOF [UMLInfra:2009] 24
2.5. Taxonomía de los diagramas estructurales y de comportamiento de UML [UMLSuper:2009] 25
2.6. Notación de una clase 26
2.7. Las clases definidas en el metamodelo del diagrama de clases (adaptado de [UMLSuper:2009]) 27
2.8. Metamodelo resumido del diagrama de tiempo (adaptado de [UMLSuper:2009] y [Muneton:2008]) 32
2.9. Arquitectura del perfil MARTE [MARTE:2009] 35
2.10. Diagrama de clasificadores, instancias y propiedades del paquete Foundations [MARTE:2009] 36
2.11. Arquitectura del paquete Causality [MARTE:2009] 37
2.12. Estructura y dependencias del paquete NFPs [MARTE:2009] 39
2.13. Estructura del modelo del dominio del tiempo [MARTE:2009] 41
2.14. Modelo de tiempo básico [MARTE:2009] 41
2.15. Modelo del tiempo con múltiples bases de tiempo [MARTE:2009] 42
2.16. RtUnit y pPUnit del paquete HLAM [MARTE:2009] 45
2.17. Estructura del marco de trabajo para modelar recursos software [MARTE:2009] 46
2.18. Dependencias del paquete GQAM [MARTE:2009] 47
2.19. Modelo del dominio para el análisis del desempeño (adaptado de [MARTE:2009]) . . 49
2.20. Estructura del marco de trabajo de VSL [MARTE:2009] 52
2.21. Extensión UML para definición de tipos de datos y variables [MARTE:2009] 54

3.1. Topología de la WPAN del sistema de vigilancia del campamento militar 57
3.2. Diagrama de procesos del sistema de vigilancia del campamento militar 58
3.3. Actividades realizadas en el marco de trabajo basado en requisitos no funcionales . . 58
3.4. SIG del sistema de vigilancia del campamento militar 61

3.5. Perfil SIG [Supakkul:2005] 63
3.6. Perfil Procedure [Supakkul:2005] 64
3.7. Porción del diagrama de clases para el SIG de la WPAN 65
3.8. Continuación del diagrama de clases para el SIG de la WPAN 66
3.9. Diagrama de casos de uso para el coordinador de la WPAN 67
3.10. Diagrama de casos de uso para los dispositivos finales que incluyen sensores y detectores de presencia de la WPAN 67
3.11. Diagrama de casos de uso para el dispositivo final que controla la alarma de la WPAN 68
3.12. Diagrama de actividades UML de la WPAN 69
3.13. Diagrama de clases UML de la WPAN 70
3.14. Diagrama de secuencia del Coordinador de la WPAN 71
3.15. Diagrama de secuencia del dispositivo final de la WPAN 73
3.16. Diagrama de tiempo del Coordinador de la WPAN 74
3.17. Diagrama de despliegue de la WPAN 75
3.18. Diagrama de estados del Coordinador de la WPAN 78
3.19. Diagrama de despliegue de la WPAN utilizando MARTE 80
3.20. Diagrama de actividades de la WPAN utilizando MARTE 81
3.21. Diagrama de secuencia del Coordinador de la WPAN utilizando MARTE 83
3.22. Diagrama de clases para representar el NFR tiempo de respuesta 85
3.23. SIG detallado para el NFR tiempo de respuesta del Coordinador 86
3.24. Integración del NFR tiempo de respuesta y el diagrama de clases 86
3.25. Metamodelo para los NFRs con conocimento del dominio (adaptado de [Jingbai:2008]) 87
3.26. Instancia de NFRs basado en una ontología del dominio para la WPAN 88
3.27. Conjunto de aspectos de DERAF para el manejo de NFRs (adaptado de [Wehrmeister:2007]) 90
3.28. Casos de uso para la WPAN con estereotipos de NFRs 90
3.29. Vinculación de los JPDD con un aspecto y su adaptación 91
3.30. Diagrama de estados del modelo UML extendido con el aspecto tiempo de respuesta 93
3.31. Estructura del modelo de datos para los NFRs [Fei:2007] 94

5.1. Actividades realizadas en el marco de UN-METODO con las modificaciones propuestas 104
5.2. Actividades realizadas en el marco del método propuesto durante la fase del esquema conceptual 105
5.3. Propuesta de modificación del metamodelo del diagrama de clases (adaptado de [UMLSuper:2009]) 107
5.4. Extensiones UML para los estereotipos de nivel superior del perfil GQAM [MARTE:2009] 108
5.5. Propuesta de modificación del metamodelo para la construcción de expresiones con VSL (adaptado de [MARTE:2009]) 109
5.6. Clase UML con y sin representación gráfica 110
5.7. Clase UML con la modificación del metamodelo y sin representación gráfica 111

5.8. Clase UML con una sintaxis combinada e incluyendo la modificación del metamodelo 112
5.9. Sintáxis para la representación de propiedades de las operaciones de una clase en el diagrama de clases de UML 114

6.1. Diagrama de clases UML de la WPAN sin la especificación de los NFRs 116
6.2. Diagrama de clases UML de la WPAN con la especificación de los NFRs 118
6.3. Diagrama de despliegue del gateway Ethernet-ZigBee [Serna:2010a] 119
6.4. Diagrama de clases del gateway Ethernet-ZigBee sin la especificación del NFR [Serna:2010a] 120
6.5. Diagrama de secuencia del gateway Ethernet-ZigBee [Serna:2010a] 121
6.6. Diagrama de clases del gateway Ethernet-ZigBee con la especificación del NFR 122

A.1. Certificado como ponente en ROPEC'2010 130
A.2. Primera página del artículo publicado en las memorias de ROPEC'2010 131
A.3. Tabla de contenido de la revista Iteckne Vol. 7, No. 1, Julio 2010 132
A.4. Primera página del artículo publicado en la revista Iteckne Vol. 7, No. 1, Julio 2010 . 133

Índice de tablas

2.1. Equivalencia entre los elementos del perfil MARTE en la vista del dominio y en la representación UML 48
2.2. Propiedades útiles para analizar el desempeño (adaptada de [MARTE:2009]) 50

3.1. Tabla explicativa de los procesos de la WPAN 59

Acrónimos y siglas

API	Application Programming Interface
CDM	Custom Development Method
CPU	Central Processor Unit
CTL	Computational Tree Logic
DERAF	Distributed Embedded Real-time Aspects Framework
DERTS	Distributed Embedded Real-Time Systems
FDAF	Formal Design Analysis Framework
FDD	Feature Driven Development
FP	Factor de Potencia
FR	Functional Requirement
FRIDA	From RequIrements to Design using Aspects
GCM	General Component Model
GPS	Global Positioning System
GQAM	Generic Quantitative Analysis Modeling
GRM	Generic Resource Modeling
HLAM	High-Level Application Modeling
IDE	Integrated Development Environment
ISO	International Organization for Standardization
JPDD	Join Point Designation Diagrams
KAOS	Knowledge Acquisition autOmated Specification
LEL	Language Extended Lexicon
MARTE	Modeling and Analysis of Real-Time Embedded Systems
MOF	Meta-Object Facility
NFPs	Non-functional Properties
NFR	Non-Functional Requirement
OCL	Object Constraint Language
ODM	Ontology Definition Metamodel
OTL	Object Temporal Logic
OWL	Web Ontology Language
OWL-DL	Web Ontology Language Description Logic

PAM Performance Analysis Modeling
QoS Quality of Service
RE Requirements Engineering
RT Real Time
RT/E Real Time Embedded
RTOS Real Time Operating System
RUP Rational Unified Process
SAM Schedulability Analysis Modeling
SCI Serial Communication Interface
SIG Softgoal Interdependency Graphs
SPI Serial Peripheral Interface
SQL Structure Query Language
UML Unified Modeling Language
UML-RT Unified Modeling Language - Real Time
VSL Value Specification Language
WPAN Wireless Personal Area Network
XMI XML Metadata Interchange
XML eXtensible Markup Language
XP eXtreme Programming

Resumen

La especificación de requisitos no funcionales en sistemas de software, ha presentado múltiples retos a académicos investigadores interesados en el tema. Las crecientes exigencias de los sistemas en características relacionadas con requisitos no funcionales – NFRs, como desempeño, seguridad, escalabilidad, entre otros, ha permitido diversos enfoques a la hora de construir los planos del software del sistema deseado por el cliente.

Esta tesis, se enmarca dentro de la especificación formal de requisitos temporales no funcionales, utilizando el lenguaje de modelado unificado – UML para la construcción de los planos del software. La tesis no trata con requisitos funcionales – RFs, ni NFRs que no tengan que ver con tiempo. La especificación se hace sobre diagramas de clase UML, no trabaja con otros diagramas ni otros lenguajes de modelado. Se apoya en métodos existentes durante las primeras etapas de desarrollo, lo que le permite elicitar los requisitos e ir llevándolos de manera consistente a través de todos los diagramas del modelo, hasta llegar a un nuevo diagrama de clases. Este nuevo diagrama de clases relaciona elementos que originalmente no está permitido en los diagramas de clase tradicionales y exige la especificación textual del NFR para darle formalidad, logrando con esto una mayor expresividad y permitiendo tomar decisiones de implementación que antes no era posible en esta etapa del desarrollo.

Para lograr representar y formalizar los NFRs, es necesario realizar una variante a la semántica del diagrama de clases de UML y al diagrama de expresiones del lenguaje de especificación de valores – VSL, permitiendo relacionar metaclases que antes no lo estaban, por ejemplo `Property` y `Operation`, y definir contextos sobre propiedades de operaciones para las nuevas expresiones. Igualmente, se introduce una nueva simbología para esta nueva representación.

Este es el trabajo de final de carrera de la maestría en ingeniería de sistemas con énfasis en ingeniería de software, de la Facultad de Minas de la Universidad Nacional de Colombia.

Medellín, *Sergio*
Marzo 2011 *Serna*

Abstract

Specifying non-functional requirements in software systems, has presented many challenges to academic researchers interested in the subject. The growing demands on system characteristics related to non-functional requirements – NFRs, like performance, security, scalability, among others, allowed different approaches in building the software plans of the system desired by the customer.
This thesis, is part of the formal specification of non-functional timing requirements using the unified modeling language – UML for building software plans. The thesis does not deal with functional requirements –RFs and NFRs that have nothing to do with time. The specification is made on UML class diagrams, does not work with other diagrams or other modeling languages. It builds on existing methods in the early stages of development, allowing you to elicit requirements and be taking them consistently throughout all the diagrams of the model, until a new class diagram. This new class diagram related items not originally permitted in traditional class diagrams and textual specification requires NFR to give formality, thus achieving greater expressivity and allowing take decisions that previously was not possible in this stage of development.
To ensure and formalize represent NFRs necessary to perform a variant of the semantics of UML class diagram and the expressions diagram of the value specification language – VSL, allowing metaclasses link before were not, e.g. `Property` and `Operation`, and define context on operations and property for the new expressions. Also introduces a new symbols for this representation.

This is the thesis in systems engineering with emphasis on software engineering in the School of Minas, Universidad Nacional of Colombia.

Medellín, *Sergio*
March 2011 *Serna*

Capítulo 1
Introducción

Resumen En este capítulo se presentan los sistemas embebidos de tiempo real y su importancia actual, se hace una descripción general de los trabajos previos en el área y se plantea el objetivo del trabajo. Los trabajos previos sólo se mencionan, ya que en el capítulo 3 se realiza una descripción detallada de los mismos. Finalmente, se relacionan los trabajos presentados en eventos y revistas indexadas como producto de las ideas desarrolladas en esta tesis.

Un aspecto crucial a la hora de crear software de calidad, es escribir claras especificaciones. Lastimosamente, a pesar de que la mayoría de los programadores reconocen la necesidad de escribir especificaciones, pocos lo hacen. Esto tiene que ver en muchos casos con la falsa creencia de que no escribir especificaciones "ahorra tiempo", y hacerlo "constituye un lujo que no representa beneficios para nadie, ni para el programador, ni el cliente, ni el usuario del sistema". Las especificaciones son esenciales en cualquier sistema no trivial [Hull:2005], e incluso en aquellos sistemas poco complejos juegan un papel muy importante a medida que éste crece. De hecho, no escribir especificaciones es un riesgo innecesario al que se somete el desarrollador y la compañía de software. Quienes se enfrentan al código, sin haberlo especificado, generalmente son tremendamente improductivos, escribiendo programas mediocres de muy baja calidad, lo que indudablemente incrementa los riesgos del proyecto.

El objetivo principal de escribir las especificaciones es poder diseñar el programa, ya que una descripción minuciosa de cómo deben funcionar las cosas obligará a realizar un verdadero diseño. La comunicación se mejora dramáticamente al tener escritas las especificaciones, ya que sólo es necesario, por una vez, informar cómo funciona el programa. Basta con que todos los miembros del equipo lean las especificaciones y ya sabrán cómo utilizarlo, si no se tiene la especificación, la comunicación pasa a ser ad hoc.

Igualmente, sin una especificación detallada, es imposible hacer una planificación temporal. No tener un plan de tiempos quizás "esté bien" para pequeños proyectos, pero resulta catastrófico para cualquier negocio real, ya que es necesario saber cuánto tiempo van a llevar las cosas, porque desarrollar un proyecto cuesta dinero. Escribir una especificación permite agilizar la toma de decisiones de diseño, desde las más complejas hasta las más simples.

Hay muchas cosas que es necesario especificar y documentar en un proyecto software, pero todo debe comenzar desde el cliente. Es necesario conocer los requisitos que el cliente le exige al proyecto,

requisitos funcionales del sistema, ya que esto obliga al diseñador a ampliar su espectro e indagar más sobre el problema, lo que facilitará encontrar la solución. Encontrar la solución le impondrá restricciones al producto ó al desarrollo del proyecto, lo que dirige la atención del equipo de trabajo a los denominados requisitos no funcionales.

Los requisitos funcionales definen lo que hace el sistema o describen la naturaleza de las interacciones entre los componentes y su ambiente. Estas son definiciones algo vagas, ya que un sistema podría hacer muchas cosas que probablemente un usuario ni siquiera se imagina que podría hacer. Los requisitos funcionales son cosas requeridas, que deben ser deseadas por el cliente y pensadas por el diseñador [Burge:2002]. Por ejemplo, en un sistema de control de tráfico aéreo, un requisito funcional puede ser que la posición del avión se debe divulgar con una exactitud de por lo menos dos kilómetros. El requisito no especifica cómo alcanzar esa exactitud, solamente que la exactitud es necesaria. Otra característica importante del requisito es que éste debe ser "testeable"[1].

Por otra parte, los requisitos no funcionales son los requisitos que imponen restricciones sobre el producto que se está desarrollando[2], sobre el proceso de desarrollo[3], o se especifica por medio de restricciones externas que el producto o el proceso deben cumplir[4]. Puesto que la distinción entre los requisitos funcionales y no funcionales puede causar confusión, hay algunas características de los requisitos no funcionales que se utilizan para distinguirlos de los requisitos funcionales: los requisitos no funcionales se centran en cómo el software debe realizar algo en lugar de centrarse en lo que hace el software; los requisitos no funcionales pueden agrupar requisitos muy generales que se aplican a prácticamente cualquier sistema, como por ejemplo aspectos de seguridad; los requisitos no funcionales no se pueden satisfacer si no se mira el sistema como un todo, esto se aplica particularmente cuando se trata de desempeño; los requisitos no funcionales típicamente no se satisfacen por completo, normalmente se suplen dentro de ciertos límites [Chung:2000].

Aunque los requisitos no funcionales son cruciales para el éxito del desarrollo del sistema, raramente son analizados. Por supuesto hay excepciones, por ejemplo, hay una clase de problemas donde algunos de los requisitos relacionan la funcionalidad del sistema con el tiempo del mundo real, el cual puede ser medido en unidades que van desde los microsegundos hasta las horas. Para estos sistemas, la línea de tiempo es tan importante como la funcionalidad y por lo tanto este tipo de restricciones de tiempo real necesitan ser analizadas cuidadosamente [UMLPSPT:2005]. Como ejemplo tenemos el servicio de canje de cheques locales que un banco presta a sus clientes, indicándoles que en dos días podrán disponer del dinero en su cuenta. Este servicio relaciona un requisito funcional, como es el canje de cheques locales, con el tiempo del mundo real, es decir, "dos días". Para que la organización pueda cumplir este requisito no funcional, debe analizar el sistema, identificar las actividades, y determinar el tiempo que toma en realizarlas. Debe también decidir qué recursos (personas, planes, herramientas, etc.) asignar a los procesos. El uso de estos recursos debe ser coordinado de tal manera que no perjudique los requisitos de tiempo de otras actividades. Si conocemos la cantidad de cheques a canjear y contamos con los recursos para ello, entonces la organización puede, ignorando fallas en algún eslabón de la cadena, garantizar la disponibilidad del dinero en la cuenta del cliente al segundo día. Si la cantidad de cheques que entran para canje es, día

[1] Es decir, es posible demostrar que el requisito se satisfizo

[2] Requisitos del producto

[3] Requisitos del proceso

[4] Requisitos externos

a día, una cantidad muy impredecible, tendrá que manejarse una predicción estadística, indicando, por ejemplo, que el dinero se encontrará disponible en la cuenta del cliente entre 48 y 96 horas después de la consignación.

Los requisitos no funcionales que se relacionan con el tiempo del mundo real son de varios tipos: en un requisito podrá exigirse un tiempo de respuesta tal que, por ejemplo, un evento "X" causará un evento "Y" dentro de 24 horas. Un requisito de desempeño puede verse como que el sistema realizará, por ejemplo, 500 ocurrencias de un evento por hora. Un requisito estadístico será algo como que el 80 % de las ocurrencias de un evento "X" causará un evento "Y" dentro de 36 horas. Todos los anteriores son requisitos de tiempo real. Un sistema que cumple alguno de estos requisitos, se dice que es un sistema de tiempo real.

En resumen, un sistema de tiempo real es un sistema donde el funcionamiento correcto depende de los resultados producidos por el sistema y el tiempo en el cual estos resultados se producen. Esto nos lleva a pensar lo importante que los requisitos no funcionales se vuelven para esta clase de sistema, y por esta razón dirigiremos nuestra atención en esta tesis a ellos, aunque no de manera exclusiva.

Existen múltiples propuestas para trabajar con NFRs, algunas se enfocan en la etapa de educción de requisitos, como aquellas que se apoyan en [Mylopoulos:1992], [Chung:1995], [Chung:2000] y [Supakkul:2005]. En este caso, a diferencia de los enfoques de diseño dirigidos por la funcionalidad, es posible utilizar un marco de trabajo basado en NFRs con el fin de dirigir el proceso de diseño completo de un sistema. Esto obliga al desarrollador a trabajar con los NFRs en mente todo el tiempo.

Otros trabajos, [Kaiya:2005] [Kaiya:2006], definen ontologias adecuadas al dominio de aplicación, esto con el fin de ayudar en el proceso de captura de requisitos software. En [Jingbai:2008] se propone un metamodelo, basado en una ontología del dominio, para NFRs de acuerdo al contexto de aplicación.

Otras metodologías se centran en definiciones de arquitecturas que soporten de forma adecuada las características transversales de los NFRs, en algunos casos la filosofía de diseño es objetual en otros es dirigida por aspectos. Algunos de los trabajos que representan la parte objetual son [Tonu:2006] [Tonu:2005], donde se propone un marco de trabajo que incorpora NFRs dentro de una notación estándar de UML sin proponer extensión alguna. En este marco de trabajo se identifican RFs y NFRs, luego se especifican los RFs y NFRs, y por último se integran unos con otros. El proceso de identificación parte de diagramas UML y técnicas de educción de requisitos. La especificación se realiza según el marco de trabajo basado en softgoal [Chung:2000] y una plantilla propuesta por los autores, las cuales son almacenados en un repositorio para posterior reutilización. En [MARTE:2009] se especifica un perfil UML para sistemas embebidos de tiempo, en el cual es posible especificar propiedades no funcionales de elementos UML que puedan mapearse a escenarios definidos en el perfil.

El otro enfoque, aspectos, se presenta en [Wehrmeister:2007], donde se define un marco de trabajo basado en aspectos para sistemas embebidos, distribuidos y de tiempo real, DERAF. DERAF propone un conjunto de aspectos con un nivel de abstracción muy alto que facilitan el trabajo con NFRs. DERAF maneja requisitos de desempeño, temporización, precisión y más.

Los lenguajes formales, como OCL, son una alternativa adicional para especificar los NFRs. En [Flake:2002a] se define un perfil UML a partir de una extensión temporal de la versión 2.0 de OCL [Flake:2002]. De esta forma, se puede utilizar OCL para especificaciones de tiempo real, ya que se

introducen operaciones temporales que permiten especificar comportamientos orientados a estados. Esta extensión razona acerca de posibles estados futuros de los objetos, donde la definición de la semántica se basa en una variante de CTL limitada en el tiempo.
Como resultado de la revisión bibliográfica, puede evidenciarse que existe una falta de consenso acerca de lo que son los NFRs y cómo deben elicitarse, documentarse y validarse, lo que ha generado múltiples debates y distintas corrientes de trabajo entre la comunidad de profesionales de la ingeniería de software. Es por esto que se hace necesaria una especificación de NFRs que permita llevarlos hasta el nivel de implementación, con el fin de garantizar su cumplimiento y por ende minimizar los riesgos del proyecto. De esta forma, en esta tesis se pretende especificar formalmente en el diagrama de clases de UML los requisitos no funcionales de un sistema, los cuales deben estar asociados a las operaciones de una clase.
Este trabajo es de carácter teórico, conceptual y metodológico, ya que plantea una manera novedosa de especificar los NFRs en el diagrama de clases de UML, esto implica extender el metamodelo del diagrama de clases y del diagrama de expresiones de VSL para abarcar propiedades de las operaciones, tema actualmente no cubierto, así como facilitar la definición de contextos del sistema en el diagrama de clases de UML de tal forma que la semántica no represente ninguna ambigüedad.
Esta tesis se organiza de la siguiente manera: en el capítulo 2, se presentan los conceptos que constituyen el marco teórico sobre el cual se soporta esta propuesta; en el capítulo 3, se presentan los antecedentes acerca de la especificación de NFRs, utilizando como ejemplo una red inalámbrica de área personal para la seguridad de un campamento militar; el capítulo 4, presenta y delimita el problema de investigación que motiva esta tesis, estableciendo el objetivo general y los específicos; en el capítulo 5, se describe el método propuesto, que incluye las propuestas de modificación al metamodelo del diagrama de clases de UML y el diagrama de expresiones de VSL, igualmente propone la sintaxis para la representación de las propiedades de las operaciones de las clases; el capítulo 6, presenta la validación del método propuesto en esta tesis utilizando el ejemplo de la red inalámbrica desarrollado en el capítulo 3, y un caso de estudio acerca de la medición remota del factor de potencia de un motor DC, utilizando un gateway Ethernet - ZigBee; por último, en el capítulo 7 se presentan las conclusiones y los posibles trabajos futuros.

1.1. Trabajos derivados de esta tesis

De la propuesta de esta tesis se realizó la ponencia “Representación de requisitos no funcionales en el diagrama de clases”, durante la celebración de la XII Reunión de Otoño de Potencia, Electrónica y Computación - ROPEC’2010 INTERNACIONAL, realizada en la ciudad de Manzanillo en México [Serna:2010]. Igualmente, se publicó un artículo en la revista Iteckne de la Universidad Santo Tomás seccional Bucaramanga [Serna:2100b], la revista actualmente se encuentra clasificada en la categoría C por Publindex (véase anexo A).
Esta tesis permitió derivar tres trabajos de grado para estudiantes de ingeniería de sistemas del Instituto Tecnológico Metropolitano: uno sobre la conceptualización del perfil UML MARTE y su utilización en sistemas RT/E que hacen parte de los proyectos de investigación del grupo de investigación GIT, utilizando la herramienta de modelado Papyrus; otro, trata sobre la implementación

del perfil UML MARTE en la herramienta de modelado Enterprise Architect^{TM}; y el último, es el estudio de algunas técnicas de análisis de desempeño para sistemas RT/E.

Capítulo 2
Marco Teórico

Resumen Este capítulo abarca los procesos involucrados en la ingeniería de requisitos, la calidad de los requisitos, y los requisitos funcionales y no funcionales. Se presenta una clasificación de requisitos no funcionales, enfocándose principalmente en aquellos que tienen que ver con el tiempo. Igualmente, se conceptualiza sobre el método de educción de requisitos utilizado en esta tesis, UNMETODO. Además, se hace una introducción al lenguaje de modelado unificado – UML, haciendo énfasis en su metamodelo y principalmente en el metamodelo del diagrama de clases. Por último, se detalla el perfil UML para modelado y análisis de sistemas embebidos de tiempo real – MARTE, el lenguaje de especificación de valores – VSL, y el lenguaje de restricción de objetos – OCL.

2.1. Ingeniería de requisitos

Si se quiere tener el producto correcto en el momento adecuado, es necesario realizar un proceso de captura de requisitos serio. Para esto, es necesario conocer los interesados[1] en el sistema y las necesidades que estos tienen. La ingeniería de requisitos – RE, se ocupa de este proceso de captura de información; a pesar de que puede verse como simple sentido común, en ocasiones no es una disciplina bien entendida y por ende se percibe difícil. Este desconocimiento, es generalmente la razón por la cual el proceso de captura de requisitos, en la mayoría de los casos, no se hace bien.
El éxito de un sistema, se mide principalmente por el grado en el cual se cumplieron los propósitos para los cuales fue concebido. Estos propósitos son tomados de los interesados, sin embargo, las necesidades de los interesados pueden ser múltiples y variadas, y con frecuencia contradictorias. Al principio, los requisitos normalmente no están claramente definidos, pueden estar restringidos por factores fuera de control o pueden estar influenciados por otros objetivos que cambian con el tiempo. Por otra parte, estos requisitos inicialmente están expresados en lenguaje natural, lo que los hace bastante imprecisos. Si no se tiene una base de requisitos estable, antes de comenzar el desarrollo del proyecto, lo más probable es que se fracase. Es como lanzarse al mar sin un destino definido y sin una carta de navegación.
Son varias las definiciones de RE que existen, tomando [Nuseibeh:2000][Zave:1997] podemos de-

[1] Los clientes, incluídos los que pagan, los usuarios, los desarrolladores, los proveedores, el negocio, entre otros

cir que:

La RE es la rama de la ingeniería de software que descubre los objetivos del mundo real de un sistema, por medio de la identificación de los interesados y sus necesidades, y documentándolos de forma adecuada para el análisis, la comunicación y la posterior implementación. También se determina por las relaciones entre la función del sistema y las restricciones que este tiene, con el fin de precisar especificaciones del comportamiento del software a medida que pasa el tiempo y que llegan nuevas versiones.

Hay varios aspectos claves en esta definición. El primero es el qué y el por qué del sistema, esto se ve reflejado en la afirmación "objetivos del mundo real". Otro elemento importante es la documentación clara y precisa, que permite un proceso de análisis riguroso así como un proceso de validación, elementos indispensables si se quiere verificar que lo construido es lo que los interesados realmente necesitaban. Por último, se consideran los aspectos de reutilización y de cambios; el primero es necesario, ya que ahorran tiempo y dinero; el segundo también es fundamental, ya que cuando un sistema cambia con el tiempo, estos cambios pueden deberse a nuevos requisitos dados por los interesados o por mejoras establecidas por los desarrolladores.
Luego de tener un acuerdo con los interesados acerca de los requisitos del sistema, y sólo después de esto, es posible planear el proyecto, administrar el riesgo, acordar pruebas de aceptación, equilibrar los diferentes elementos del sistema, y manejar el control de cambios [Hull:2005].

2.1.1. Calidad de los requisitos

No tener requisitos, tenerlos incompletos, desorganizados, ambiguos, o poco factibles tiene diversas consecuencias. Un sistema así, aunque en principio parezca trabajar bien, con seguiridad no será el sistema por el cual el cliente pagó y que los usuarios necesitan.
Para tener requisitos de buena calidad, es necesario que estos sean correctos, factibles, completos, consistentes, modificables, necesarios, estén priorizados, no sean ambiguos, tengan trazabilidad y puedan verificarse [IEEE830:1998] [Wiegers:1999]. Por lo anterior, podemos afirmar:

La calidad de un requisito habla de la capacidad que tiene este para contribuir a alcanzar los propósitos del sistema, esto es, se refiere a aquello que provee el requisito para contribuir a la satisfacción del cliente, teniendo en cuenta todas las necesidades de los interesados.

Para aumentar las posibilidades de éxito durante el desarrollo de un sistema software, se debe describir exactamente la funcionalidad que el requisito entregará, de esta manera es posible descartar aquellos que entren en conflicto con un requisito del sistema. Luego de establecer cuales son los requisitos correctos, estos se deben evaluar con el fin de determinar la factibilidad de su implementación, ya que las limitaciones de la tecnología y los costos pueden hacer inviable su cumplimiento. En etapas más avanzadas del desarrollo del proyecto, es necesario identificar el origen de cada requisito, esto es, un caso de uso, un requisito del sistema, una regulación, u otro, ya que si no se

conoce su origen, es altamente probable que el requisito no sea necesario. Además, se hace indispensable priorizar los requisitos, ya que de esta manera es posible tener políticas claras acerca de como reaccionar a nuevos requisitos, a recortes presupuestales, a retrasos en la programación o a la necesidad de prescindir de un miembro del equipo de trabajo.
El valor de un requisito, lo da la importancia que el cliente le asigne, el costo de implementarlo y el riesgo técnico de llevarlo a cabo. Una descripción no ambigua de un requisito, permite que el lector entienda una y solo una cosa. De esta manera, se garantiza que múltiples lectores entiendan exactamente lo mismo. Por último, el requisito debe ser verificable, ya que de otra forma, determinar si fue correctamente implementado es sólo cuestión de opinión.

2.1.2. Actividades de la ingeniería de requisitos

En la sección 2.1 definimos la RE, y de acuerdo a esta definición son varias las actividades que los ingenieros deben desarrollar. Con el fin de identificar los interesados y sus necesidades, y estas últimas documentarlas para su posterior análisis, comunicación e implementación se requiere desarrollar las siguientes actividades [Nuseibeh:2000]: educción, modelado, comunicación, aceptación y evolución.
La educción de requisitos consiste en la captura de los mismos, obteniendo, luego de la interpretación, análisis, modelado y validación, el conjunto de requisitos necesario para el desarrollo del sistema.
Es aquí donde se delimita el problema, se conoce exactamente qué es lo que se va a solucionar y de esta forma, es posible definir el alcance. Este alcance, permite identificar correctamente los interesados y las clases de usuarios, los objetivos y las tareas, y los casos de uso y los escenarios del sistema.
Son variadas las técnicas que pueden utilizarse para la educción de requisitos, todo depende del tiempo, presupuesto y el tipo de información que se desea capturar. Las técnicas van desde los tradicionales cuestionarios dirigidos a los usuarios, pasando por técnicas grupales, como lluvia de ideas, prototipado, KAOS [VanLamsweerde:1998] y el marco de trabajo para requisitos no funcionales – NFR [Chung:2000], entre otros.
Para los complejos sistemas software actuales, la cantidad de requisitos, las relaciones entre ellos, y las restricciones de cada uno, hacen imposible una comprensión completa, clara y total del mismo. Para enfrentar este reto y salir avante, se requiere modelar el sistema con el detalle apropiado de acuerdo al propósito que se tiene. Son varias las definiciones de modelo que existen en la literatura, tomando a [Hull:2005] [Jacobson:2000] podemos decir:

Un modelo es una abstracción semanticamente cerrada de un sistema, que se enfoca deliberadamente, en los aspectos del sistema que son de interés, mientras excluye otros que, aunque importantes, pueden distraer, ya que no son relevantes para la situación en particular.

De esta manera, se pueden agrupar cantidades reducidas de información en uno o varios modelos, convirtiendo el sistema en una colección de modelos relacionados entre sí. Esta agrupación de

información en pequeños modelos permite el análisis y procesamiento del sistema, facilitando la aplicación de técnicas específicas pertinentes a los aspectos bajo estudio.
¿Qué se debe modelar?, esa es la pregunta que ha permitido la proliferación de múltiples métodos de desarrollo de software, algunos enfocados en los artefactos de gestión, otros en los artefactos de ingeniería. Sin embargo, de ambos tipos de artefactos se extrae información útil.
Otro tipo de sistema importante son los sistemas de información que producen o necesitan grandes volúmenes de datos. Para que estos datos sean correctamente entendidos, manipulados y administrados, es necesaria una representación que defina y analice los requisitos de los datos, de tal forma que soporten adecuadamente los procesos de negocio de una organización. Existen básicamente tres modelos de datos: conceptual, lógico y físico.
Los modelos conceptual y lógico se utilizan para explorar conceptos del dominio y sus relaciones, esto es, los objetos del mundo real en el dominio de interés y cómo interactuan entre ellos [Arango:2006].
El modelo de datos físicos, describe el esquema interno de una base de datos, esto es, sus tablas, columnas y las relaciones entre las tablas. El diagrama entidad relación es lo más utilizado para el modelo físico de datos.
Modelar la dinámica del sistema, es clave para la comprensión de su funcionalidad por parte de los interesados, de esta manera es posible, partiendo de un modelo y una entrada, obtener la salida deseada. Existen múltiples métodos para modelar el comportamiento, algunos de ellos formales, otros no tanto. Entre los más comunes se encuentran las especificaciones axiomáticas, las especificaciones de transición de estados, las especificaciones de modelos matemáticos, las especificaciones algebraicas, las especificaciones basadas en lógica temporal y las especificaciones concurrentes [Sommerville:2007].
Finalmente, el modelo donde se incluyan correctamente los NFRs y resulte lo suficientemente expresivo, de tal forma que se permita analizarlos y medirlos, no es fácil de lograr. Esto se debe fundamentalmente a que los NFRs son requisitos muy generales, que imponen restricciones a las soluciones de acuerdo a las posibilidades del mundo real. Al ser requisitos transversales, no es posible verificar su funcionamiento en componentes individuales. Actualmente, existen varios enfoques para tratar los NFRs, entre ellos están [Chung:2000] [Xu:2005] [Botella:2001].
La tercera actividad de la RE es la comunicación de los requisitos, lo cual debe realizarse de manera efectiva entre los diferentes interesados en el sistema. Para la comunicación se tienen una diversidad de lenguajes, desde aquellos formales que utilizan algebra o lógica matemática [Milner:1982] [Smith:1999], pasando por los semiformales que utilizan notaciones gráficas [UMLSuper:2009], hasta llegar a los completamente informales, basados en el lenguaje natural.
La búsqueda de un acuerdo con los interesados con el fin de que estos acepten los requisitos capturados es la cuarta actividad a desarrollar. Esto se puede lograr realizando un proceso de validación del requisito, verificando su sintaxis y consistencia, lo que permitirá que los interesados se convenzan de que el requisito si satisface sus deseos iniciales, sobre todo en escenarios con requisitos en conflicto.
Por último, el control de versiones y la administración de cambios [Robbes:2008] es una actividad fundamental en la RE, ya que permite adicionar nuevos requisitos, borrar otros que ya no son pertinentes o simplemente corregir errores. De esta manera, es posible establecer la consistencia de los requisitos.

2.2. Requisitos de software

El desarrollo de un sistema se guía de acuerdo a las condiciones impuestas por los clientes. Estas condiciones son un reflejo de las necesidades que se espera el sistema cubra, y es lo mínimo por lo que el cliente pagará. La RE captura estas necesidades del cliente y de acuerdo a ellas guía el proceso de desarrollo hasta la obtención del producto final.
La RE se ha enfocado fundamentalmente en los llamados FRs, mientras que los NFRs se han tratado de una forma distinta.
Para justificar esta separación y utilizarla en nuestro sistema es necesario responder varias preguntas [Burge:2002]:

- ¿Qué significa realmente funcional?
- ¿Son los NFRs realmente no funcionales o son una forma de RFs?
- ¿Son los NFRs realmente requisitos?
- ¿Pueden los NFRs ser representados como FRs?

Podríamos hacer un profundo análisis acerca de todos estos conceptos y sus diferentes implicaciones de acuerdo a la rigurosidad con la que se quiera abordar el tema, pero para nuestros propósitos basta con poner en claro que es una función, y cómo podemos relacionarla con los FRs. Tomando [Chandrasekaran:2000], podemos afirmar:

La función, se define como un conjunto de efectos que una entidad tiene sobre el ambiente. Estos efectos deben ser deseados por algún agente. Las funciones son utilizadas por los agentes en algún ambiente con el fin de lograr objetivos. Si cuando una entidad se despliega en un ambiente se satisfacen en este un conjunto de restricciones de comportamiento, se dice que la entidad juega un rol en el ambiente, si este rol es deseable por algún agente, entonces la entidad cumple una función.

Para establecer que elementos cumplen funciones, y quizás satisfagan FRs, consideremos un robot con desplazamiento autónomo y representemos su funcionamiento. Por ejemplo, el robot debe estar en capacidad de detectar objetos durante su movimiento y establecer cuando estos se encuentren a una distancia menor que la distancia límite. Esto sirve para propósitos de tomar acciones con el fin de que se evite un choque. La representación funcional para este comportamiento sería la siguiente:

```
FUNCION (Control Obstáculos) DE (Sistema de Control)
  HACER (girar(derecha) o girar(izquierda))
  SI (distancia objeto <= distancia límite objeto)
```

Una descripción de esta clase tiene implícita una serie de acciones, por ejemplo, girar a derecha implica activar un pin de salida del controlador, el cual a su vez activa una entrada del puente H que maneja el motor DC y desactiva la otra, llevando uno de los transistores a corte y el otro a saturación, y energizando la armadura del motor con cierta polaridad. Esto hace fluir corriente en determinado sentido, lo cual genera un campo magnético, que junto con el que genera el devanado de campo, producen un par que es llevado por la transmisión al eje de las llantas para lograr la desviación del robot.

De esta forma, es posible identificar funcionalidad en prácticamente todas las entidades del sistema, no importa si los usuarios desean o no un comportamiento, el hecho de que este se produzca y genere un efecto sobre el ambiente, es por si solo suficiente para considerarlo funcional. Precisamente, toda esa funcionalidad junta es la que permitirá que el robot alcance los objetivos deseados por sus usuarios.

2.2.1. Requisitos funcionales

Como se describió en la sección 2.2, los FRs son aquellos que capturan la naturaleza de la interacción entre los componentes y su ambiente. Esto nos permite pensar en FRs como algunas cosas que el sistema hace y que no las solicita el cliente. Y no estamos lejos de la realidad si pensamos en el usuario, no sólo como aquel que en últimas utiliza el sistema, sino como aquellos que intervienen en los diferentes ambientes del ciclo de vida del sistema. Pensemos inicialmente en sólo tres ambientes para el sistema, seguramente hay más, el de desarrollo, el de operación y el de mantenimiento.
Cada uno de los ambientes mencionados tiene sus propios requisitos. Aquellos asociados con el ambiente de operación son los regularmente identificados como FRs, ya que son aquellos servicios los que se quiere que el sistema le proporcione a sus usuarios.
En el ambiente de desarrollo, se pueden tener requisitos que tengan que ver con la metodología, las herramientas, el equipo de trabajo o la plataforma hardware. Todos estos requisitos, la mayoría de las veces, son transparentes para el usuario final de la aplicación. Sólo en muy pocas ocasiones estos requisitos los exige el cliente.
Los requisitos del ambiente de mantenimiento, tienen que ver con hechos que se producirán luego de que el sistema ha sido desplegado. Uno de ellos podría ser la capacidad de que el sistema crezca con el fin de atender más flujo de trabajo, pero sin perder calidad en el servicio que presta. Esto es lo que comúnmente se conoce como escalabilidad.
Definitivamente, si consideramos todos los ambientes ya mencionados, los requisitos a cualquier nivel tendrán interacción con el ambiente. Y según [Chandrasekaran:2000] y la definición dada en la sección 2.2, no importa si el requisito le resulta extraño al usuario final, como por ejemplo, que el sistema sea portable, fácil de mantener, fácil de probar o eficiente, el simple hecho de interactuar con el ambiente de una forma esperada por un usuario de ese ambiente, lo hará funcional.
Entonces donde quedan los NFRs, ¿por qué se habla de NFRs y se dan clasificaciones si según lo visto cualquiera de ellos puede catalogarse como FRs? Esto lo resolveremos en la sección 2.2.2.
Ahora vamos a concentrarnos en obtener una definición de FRs [Xu:2005] [Sommerville:2007] [Robertson:1999] [Chung:2000] [IEEE:1990]:

Los FRs capturan el comportamiento esperado del sistema, en términos de los servicios o tareas que se requiere este realice. Igualmente, especifican una función que el sistema debe ser capaz de hacer, esto define el comportamiento del sistema fundamentalmente como el proceso de transformación que el hardware y el software realizan sobre las entradas para producir las salidas.

La definición deja abierta bastantes preguntas, sobretodo acerca del ambiente y los usuarios, y la interacción entre estos, pero para nuestro propósito, esta definición es suficientemente buena.

2.2.2. Requisitos no funcionales

Un tema para un debate candente entre ingenieros de software, son los NFRs. Aunque el término lleva bastante tiempo dentro del lenguaje de los ingenieros de software, aun no se tiene un consenso respecto a la naturaleza de esta clase de requisitos y como documentarlos. En esta sección se explorará este concepto, pero no discutiremos los diferentes problemas que se tienen con su definición, clasificación y representación, esto se tratará en el capítulo 4.
Los NFRs son requisitos muy generales, que imponen restricciones a las soluciones de acuerdo a las posibilidades del mundo real. Por ejemplo, un robot puede tener como NFR confiabilidad en el desplazamiento, lo cual se puede traducir en un FR para el sistema de control: posicionamiento del robot dentro de un área previamente definida, de tal manera que no se le permita abandonarla. Para esto, existen varias soluciones, pero seguramente soluciones que tomen mucho tiempo de procesamiento serán rechazadas, ya que se corre el riesgo de rebasar los límites del desplazamiento mientras se procesa la información y se da una respuesta. Soluciones basadas en GPS tardan tiempos del orden de los segundos para entregar la latitud y longitud del robot, a parte del margen de error de la medida, lo que quizás resulte ser excesivo si se quieren tomar decisiones de control sobre los motores. De esta manera, la plataforma está siendo restringida. De los NFRs podemos afirmar [Burge:2002]:

Mientras que los FRs describen lo que el sistema o dispositivo debe hacer, los NFRs se concentran en la manera como el sistema o dispositivo debe acompañarse de tal forma que se satisfagan los FRs. Los NFRs describen atributos del sistema que contribuyen a la calidad del producto como un todo. Estos atributos no están confinados en una porción de funcionalidad del sistema. Los NFRs son muy generales y probablemente el mismo NFR tiende a ser deseado de manera diferente dependiendo del sistema. Los NFRs tienen una serie de características que los diferencias de los FRs, entre ellas están: los NFRs no describen nada que el sistema hace, por ejemplo, no son requisitos que el sistema debe ejecutar; los NFRs no se relacionan con un componente específico del sistema, ya que su naturaleza los hace transversales; los NFRs no pueden ser evaluados sin mirar el sistema como un todo.

Los NFRs son claves a la hora de buscar sistemas software con calidad, de hecho, en ocasiones a los NFRs se les denomina requisitos de calidad. No tener en cuenta los NFRs al momento del desarrollo, puede darnos como resultado software inconsistente y de pobre calidad, así como usuarios, clientes y desarrolladores insatisfechos. Corregir esta omisión generará sobrecostos, tanto en dinero como en tiempo.

2.2.2.1. Tipos de requisitos no funcionales

Hay una gran variedad de NFRs, los cuales son indispensables tener en cuenta ya que generalmente pueden ser restricciones al proceso de desarrollo, al producto o presentarse como requisitos externos. Sin embargo, se debe tener claro que no existe una definición formal o una lista completa de NFRs, ni hay un esquema de clasificación universal que satisfaga todas las necesidades de los diferentes dominios de aplicación bajo todas las posibles situaciones. A pesar de esto, existen algunos acercamientos a clasificaciones de NFRs.
Basados en [Chung:2000] y [IEEE830:1998], se pueden categorizar los NFRs en restricciones de interfaz, de desempeño, operativas, de recursos, de verificación, de documentación, de portabilidad, de seguridad, de calidad, del ciclo de vida del desarrollo del software, de confiabilidad, de mantenibilidad, económicas, entre otras.
Otra clasificación la da la ISO [ISO/IEC:1999], en la cual se clasifican los requisitos para garantizar la calidad del software, requisitos de calidad que muchos autores asocian directamente con NFRs. Por ejemplo, la ISO define seis categorias de requistos de calidad: los que tienen que ver con la funcionalidad, aquellos relacionados con la confiabilidad, los asociados a usabilidad, los de eficiencia, los de mantenibilidad, y los de portabilidad. Dentro de cada una de estas categorias existen múltiplos NFRs, algunos de los cuales son: para la funcionalidad deben considerarse la exactitud, la seguridad y la interoperabilidad; para la confiabilidad se debe hablar de la tolerancia a fallas y la recuperabilidad; para la usabilidad están la comprensibilidad y la operabilidad; por la eficiencia entran el tiempo y la utilización de recursos; para el caso de la mantenibilidad están la estabilidad y analizabilidad; por último, para la portabilidad se deben tener en cuenta la adaptabilidad, instalabilidad y coexistencia.
Como se puede inferir, son múltiples los tipos de NFRs que se han identificado y que por ende no facilita el proceso de clasificación de los mismos. Sin embargo, esta tesis se enfoca en los requisitos de desempeño, los cuales incluyen propiedades de tiempo y espacio.

2.2.2.2. Requisitos no funcionales de desempeño

Como con la mayoría de los NFRs, tratar con requisitos de desempeño no es fácil, ya que se hace necesario resolver los conflictos resultantes de la interacción entre diferentes NFRs y entre sus múltiples implementaciones.
Para especificar el desempeño de un sistema, es necesario tener claridad acerca de lo que significa buen desempeño [Lazowska:1984]:

Un sistema con buen desempeño es aquel que realiza su función en el menor tiempo posible y utilizando la menor cantidad de recursos. Para efectos prácticos, y a pesar de las diversas categorias de requisitos de desempeño que pueden existir, los desarrolladores que construyen sistemas en los cuales el desempeño es un requisito importante, se enfocan fundamentalmente en la velocidad a la cual el sistema puede realizar su función, teniendo en cuenta los recursos finitos y sus características de QoS.

La dificultad de tratar con requisitos de desempeño, radica básicamente en que estos requisitos tienen un impacto global sobre el sistema, por lo tanto, satisfacer una restricción de desempeño, no consiste en sólo “adicionar un módulo extra” al sistema, sino que es necesario considerar el requisito a través de todo el sistema, y durante todo el proceso de desarrollo.
Existen varios conceptos que hacen parte de lo que se denomina desempeño de un sistema, los cuales cada vez pueden ser subdividos en conceptos más especializados (véase figura 2.1), que finalmente pueden ser operacionalizables [Chung:2000].

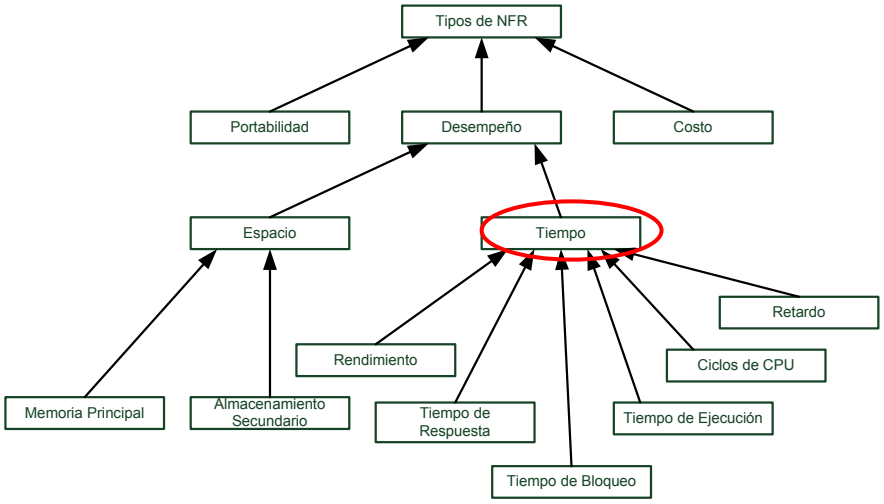

Figura 2.1: Tipos de requisitos no funcionales, destacando al tiempo como subtipo del desempeño

Para lograr el buen desempeño de un sistema software, es necesario enfocarse en los elementos que son prioridad y tratar de compesar aquellos aspectos que no pueden satisfacerse completamente. En general, para tratar los problemas de desempeño en un sistema software, un ingeniero requiere considerar, como entradas para el proceso de desarrollo del sistema, los siguientes factores: los requisitos de desempeño, los NFRs que no son de desempeño, los FRs, las prioridades del sistema y la organización, la carga de trabajo esperada para la organización o el sistema, las técnicas de desarrollo, la interacción y compensación de los NFRs, entre otros. Estas entradas al proceso de desarrollo producirán unas salidas, las cuales deben incluir por lo menos: un registro de las decisiones de desarrollo que se realizaron y las razones para tomar dichas decisiones, un indicador de los requisitos de desempeño que se lograron y en que grado, un registro de las interacciones y compensaciones entre los NFRS, las prioridades, la carga de trabajo, las decisiones tomadas, y las alternativas no seleccionadas, y una predicción del desempeño del sistema seleccionado. Una vez se tienen estos resultados se debe pasar al proceso de refinamiento de requisitos [Chung:2000], el cual permitirá llevar dichos requisitos hasta el punto de convertirlos en operacionalizables.

2.3. UN-METODO para la educción de requisitos de software

Cuando las actividades descritas en la sección 2.1.2 se agrupan dentro de un proceso sistemático se tiene un método de desarrollo software. Existen múltiples procesos de desarrollo de software, los cuales dependen del paradigma utilizado: CDM [Oracle:2000], RUP [Jacobson:2000], FDD [Coad:1999], XP [Beck:2000] entre otros. En esta sección, se estudiará un método de desarrollo software que agrupa un conjunto de diagramas característicos de diferentes paradigmas, el cual integra lo mejor de varios procesos de desarrollo.

2.3.1. Generalidades de UN-METODO

UN-METODO [Arango:2006] es un método de desarrollo de software creado en la Escuela de Sistemas de la Universidad Nacional de Colombia, el cual pretende agrupar, en un sólo proceso de desarrollo, los aspectos positivos de procesos de desarrollo reconocidos, CDM, RUP, XP, FDD, y eliminar algunos de sus aspectos negativos. Algunas de las características de UN-METODO son [Arango:2006]:

- Se genera la documentación necesaria para garantizar la trazabilidad de las piezas del software.
- Permite modificación de diagramas y artefactos diseñados en las etapas previas del método.
- Incluye diagramas de uso común, algunos de UML y CDM, y otros nuevos propuestos por los autores.
- Para llegar a la solución parte del contexto.
- Se pueden identificar fases del proceso asociadas a los entregables definidos en el método.
- Se emplean representaciones formales y semiformales para los diferentes elementos del modelo.
- El método incluye artefactos muy cercanos al lenguaje natural, como el esquema preconceptual y el modelo del dominio, y otros muy cercanos a la máquina, como los diagramas UML.
- Se define un conjunto de reglas para garantizar la consistencia entre los diferentes elementos del método (elementos de la organización, el problema y la solución).

En su mayoría, los artefactos que componen UN-METODO existen en otros métodos de desarrollo software, por lo que el aporte del método está centrado básicamente en el método mismo.
En la Figura 2.2 se presenta el diagrama de actividades de UN-METODO, en el cual se identifican cuatro grandes actividades: contexto del software, análisis del problema, propuestas de solución y esquema conceptual.
Durante la fase de la definición del contexto del software, se busca identificar la organización, sus funciones, objetivos, su vocabulario, los actores y el rol que estos juegan, con el fin de delimitar claramente el contexto de la pieza software que se piensa construir. Esta fase da como resultado la identificación de los actores, el esquema preconceptual [Zapata:2006] y el modelo del dominio. Cabe destacar que el esquema preconceptual es una creación propia de los autores de UN-METODO, y sirve como representación intermedia entre el modelo verbal y los diferentes esquemas conceptuales. El esquema preconceptual resuelve algunas ambigüedades inherentes al lenguaje natural y que por su naturaleza están presentes en el modelo verbal.

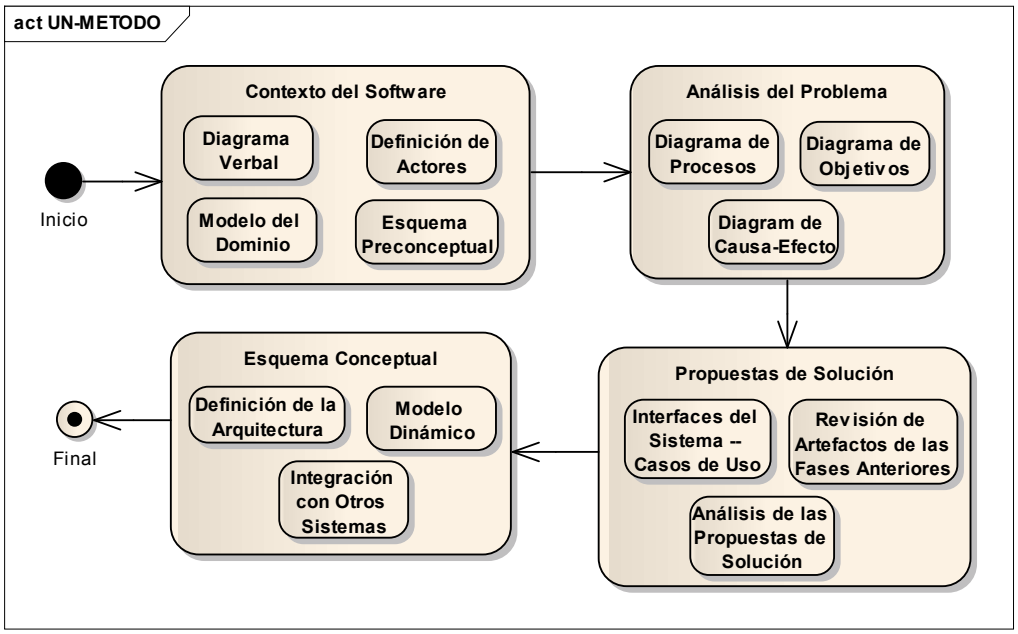

Figura 2.2: Diagrama de las actividades desarrolladas en el marco de UN-METODO (adaptado de [Arango:2006])

Durante el análisis del problema, se pretende identificar los procesos del área, cómo estos aportan a los objetivos de la organización y cuáles son los problemas a resolver. En los procesos del área es necesario identificar los involucrados, la información necesaria para llevarlos a cabo, la duración de cada proceso, la frecuencia de cada uno de ellos, el lugar donde se llevan a cabo y las regulaciones que los rigen. Esta fase arroja como entregables los diagramas de procesos [Harrington:1991], de objetivos [Dardenne:1993] y causa-efecto [Ishikawa:1986].
Para determinar la relación existente entre los procesos del área y la organización, los autores de UN-METODO crearon la tabla explicativa de los procesos. Esta tabla permite relacionar los procesos con los objetivos, los problemas y las reglas del negocio.
En la fase de propuestas de solución se describen las características generales de esta, se revisan los diferentes artefactos del sistema, y se analizan las propuestas.
Las características generales se describen por medio de la definición de las interfaces y los casos de uso. La revisión de los artefactos previos puede dar lugar a nuevos actores, nuevos roles y nuevos diagramas: procesos, objetivos y causa-efecto, entre otros. El análisis de la propuesta se centra en el costo y los factores críticos de éxito.
Luego de seleccionada la solución adecuada, se pasa al proceso de análisis y diseño detallado, denominado esquema conceptual. Esta fase se basa en UML [UMLSuper:2009], y tiene como propósito principal definir la arquitectura básica del sistema, y el comportamineto del mismo. Para este caso los diagramas UML de clases, despliegue, objetos, secuencias, estado, tiempo, actividades, entre otros, son claves. La ambigüedad en la especificación se elimina utilizando lógica de predicados en los diagramas de UML. La integración con sistemas existentes se logra por medio del uso de los diagramas de interacción de UML.

2.4. El lenguaje de modelado unificado

UML proporciona las herramientas necesarias para diseñar, analizar e implementar sistemas basados en software, así como para el modelado de negocios y procesos similares [UMLInfra:2009]. En UML, se tienen definidos múltiples conceptos que resultan útiles para el modelado de gran cantidad de sistemas, pertenecientes a los más diversos dominios de aplicación. Debido a su modularidad, no es necesario conocer todo el lenguaje con el fin de utilizarlo efectivamente en un dominio particular.
Con el fin de lograr su objetivo como lenguaje de modelado, la especificación estándar de UML contiene lo siguiente [UMLInfra:2009]:

- El metamodelo definido formalmente según MOF Meta-Object Facility [MOF:2002]. Aquí se especifica la sintáxis abstracta del lenguaje, esto es, se definen los conceptos de modelado de UML, sus atributos y relaciones, así como las reglas de combinación de estos conceptos con el fin de construir modelos UML parciales o completos
- La explicación detallada de la semántica de cada concepto de modelado UML. Esto sirve para la implementación por medio de computadores
- La notación adecuada de cada elemento del lenguaje con el fin de ser fácilmente leído por humanos, así como las reglas para combinarlos en diferentes tipos de diagramas
- La definición detallada de como las herramientas UML deben cumplir el estándar, de tal forma que se soporte el intercambio de modelos entre herramientas

Nuestro interés se centra en el metamodelo de UML, la semántica de cada elemento y su notación.

2.4.1. El metamodelo de UML

La arquitectura del metamodelo de UML se construyó pensando en cinco principios de diseño: modularidad, división en capas, organización en áreas conceptuales dentro de una misma capa, extensibilidad y reutilización.
La modularidad permite que los paquetes definidos en el metamodelo tengan una fuerte cohesión y poco acople. La división en capas del metamodelo permite aplicar, en forma consistente, la abstracción de cada capa a intereses diversos. La organización en áreas conceptuales, facilita la cohesión dentro de los paquetes y debilita el acople entre ellos. La extensibilidad, permite extender el lenguaje a nuevos dominios, por medio de la definición de perfiles. Por último, la reutilización de las librerías del metamodelo permite usarlas para definir a UML o a otros lenguajes.
Hay dos elementos claves que permiten la construcción de UML, primero que tiene un núcleo común con MOF, el paquete `Core` [UMLInfra:2009], y segundo que UML se define como un modelo de MOF, esto es, MOF hace las veces de metamodelo de UML. Tener un núcleo común permite que los elementos del modelo sean compartidos entre MOF y UML, hacer a UML un modelo de MOF significa que cada elemento del modelo de UML es una instancia de exactamente un elemento de MOF (véase Figura 2.3).

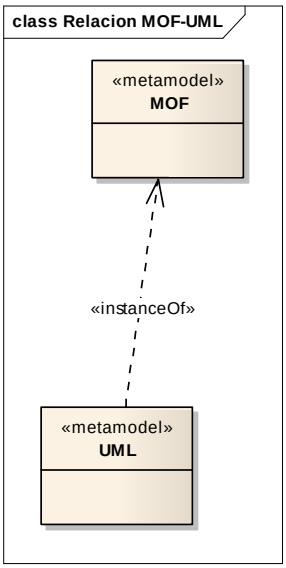

Figura 2.3: Metaniveles diferentes de UML y MOF [UMLInfra:2009]

Existe una jerarquía de cuatro capas, en la cual MOF es el meta-metamodelo y UML es el metamodelo. Los otros dos elementos son la especificación del usuario, o modelo, y los objetos del modelo (véase Figura 2.4).

Nótese que `Class`, `Instance` y `Attribute` en la capa UML son instancias del elemento `Class` de la capa MOF. Igualmente, los elementos del modelo, `Nodo`, su instancia y la relación de dependencia entre ambos, que aquí se estereotipa como `snapshot`, son instancias de los elementos `Class`, `Instance` y `Classifier` de la capa UML. Por último, `tNodo`, objeto en tiempo de ejecución, es una instancia de la clase `Nodo` de la capa del modelo de usuario. De esta forma, podemos afirmar que `Class` de la capa MOF es una meta-metaclase, mientras que `Class`, `Instance` y `Attribute` de la capa UML son metaclase, `Nodo` es una clase y `tNodo` un objeto.
Aquí se ilustra un metamodelo de cuatro capas (M3 a M0), pero no quiere decir que el metamodelo deba restringirse a sólo cuatro niveles, es posible tener tantos niveles como sea necesario. De esta manera, la capa superior puede considerarse el metamodelo de la capa inferior, la que en este caso será el modelo, igualmente esta última será el metamodelo de la capa inmediatamente debajo de esta. La recursividad permite, teóricamente, un número infinito de capas. Sin embargo, ¿cómo saber cuántas capas se deben definir? La clave está en una característica específica que tiene el metamodelamiento, y es el hecho de que permite definir lenguajes reflexivos[2]. La `InfrastructureLibrary` [UMLInfra:2009] es un ejemplo de un lenguaje reflexivo, ya que contiene todas las metaclases requeridas para definirse a si mismo. MOF está basado en `InfrastructureLibrary`, lo que lo hace reflexivo, esto es, puede definirse a si mismo. Es por esta razón que MOF es la última o primera capa del metamodelo.

[2] Los lenguajes reflexivos son aquellos que pueden utilizarse para definirse a si mismos

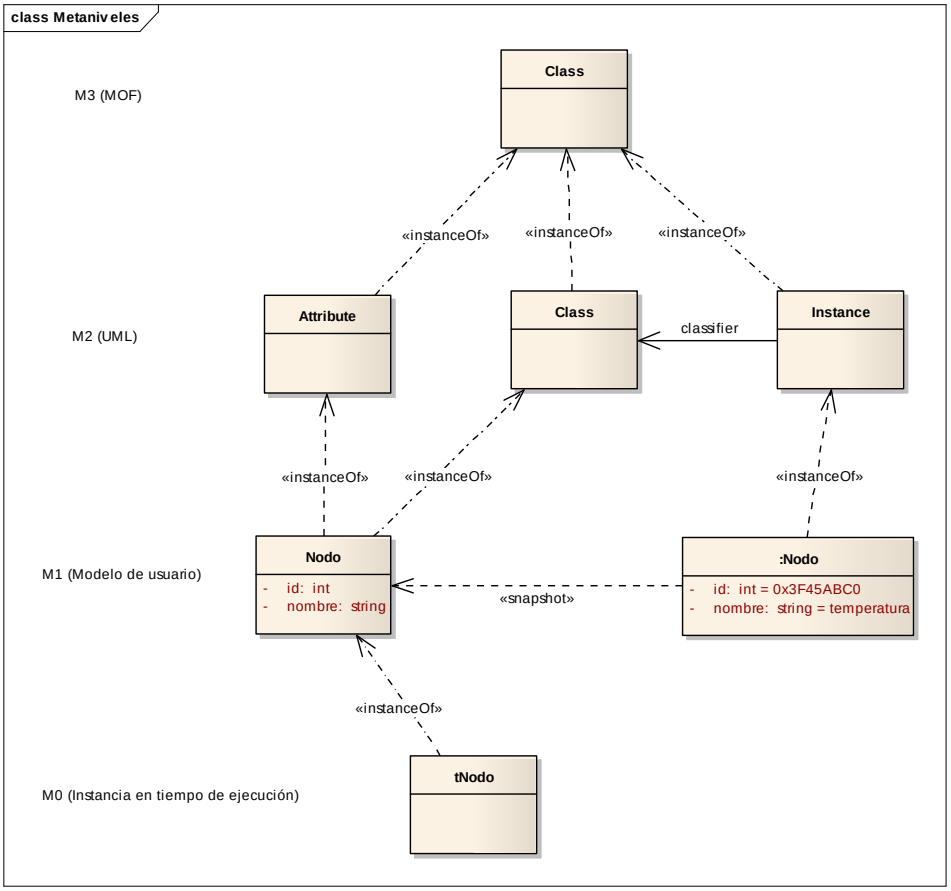

Figura 2.4: Metamodelo de 4 capas para UML y MOF [UMLInfra:2009]

2.4.2. Diagramas UML

Existen dos tipos básicos de diagramas en UML: los diagramas estructurales y los diagramas de comportamiento [UMLSuper:2009]. Los diagramas estructurales muestran la estructura estática del sistema que se quiere modelar, enfocándose, independientemente del tiempo, en los elementos del sistema. Los diagramas de comportamiento describen el comportamiento de un sistema o proceso de negocio, esta descripción está dada por medio de los diagramas de actividades, de estado, casos de uso, comunicación, secuencia, entre otros (véase Figura 2.5).

2.4.2.1. Diagramas estructurales

Los diagramas estructurales muestran la estructura estática de los objetos en el sistema. Los elementos de un diagrama estructural representan conceptos que son útiles a la aplicación en particular, y pueden incluir abstracciones o conceptos del mundo real. Por ejemplo, un diagrama estructural para una red de sensores puede incluir clasificadores que representen nodos de la red, sensores,

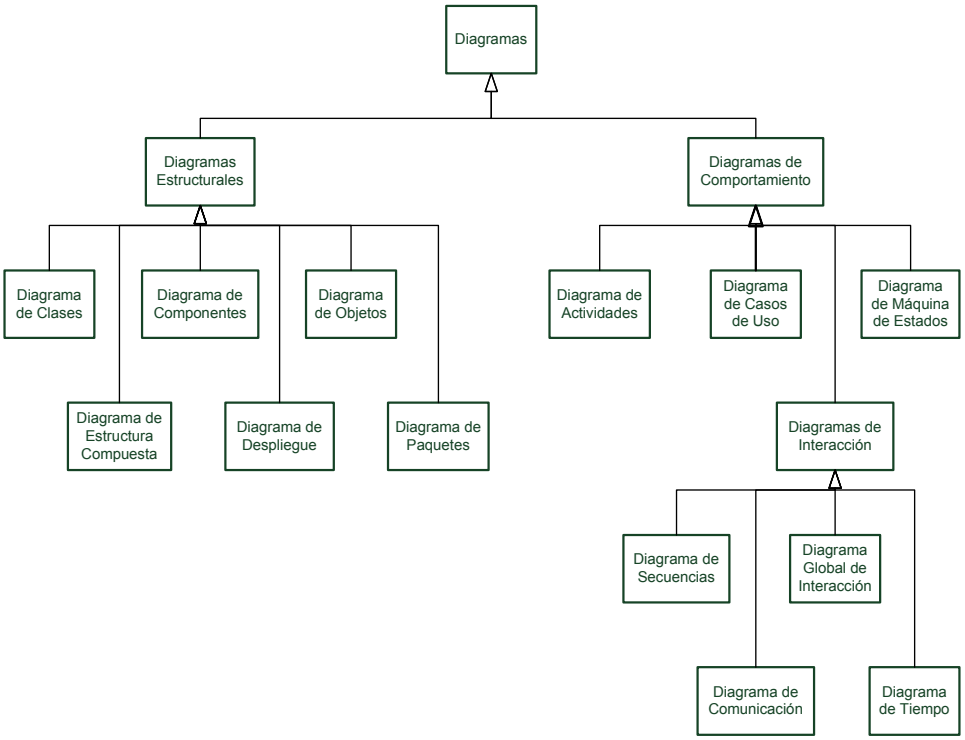

Figura 2.5: Taxonomía de los diagramas estructurales y de comportamiento de UML [UMLSuper:2009]

actuadores, algoritmos de asignación de direcciones de red, algoritmos para encriptación de información, entre otros.

Diagrama de clases

El diagrama de clases es uno de los diagramas estructurales de UML, y es el pilar del diseño y análisis orientada a objetos. Con este diagrama se describen las diferentes clases de un sistema, las interrelaciones entre ellas, sus propiedades y operaciones.
En UML una clase se representa por medio de un rectángulo con tres compartimientos. El compartimiento superior contiene el nombre de la clase, el compartimiento del medio la lista de propiedades de la clase y el compartimiento inferior la lista de operaciones de la clase (véase figura 2.6).

Las propiedades son elementos tipados que representan los atributos de una clase, estos elementos son opcionales, y se presentan de la forma:

```
nombre atributo: tipo = valor por defecto
```

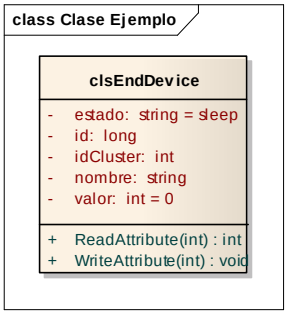

Figura 2.6: Notación de una clase

Cuando se declara un atributo en una clase que hace parte de un diagrama de clases de negocio, normalmente el tipo del atributo corresponde a una unidad que tiene sentido para el lector del diagrama, y no necesariamente es un tipo implementado en algún lenguaje de programación. Sin embargo, si se quiere generar código a partir del diagrama de clases, es necesario que el tipo del atributo se corresponda con uno definido en el lenguaje de programación en que se quiera implementar, o como mínimo el tipo debe implementarse en el sistema. Luego de declarado el atributo, el valor por defecto es opcional.
Una operación es la especificación de una transformación o consulta ejecutada por un objeto. Las operaciones tienen un nombre y una lista de parámetros, son opcionales y se representan de la forma:

```
nombre operación(parámetros): tipo valor retorno
```

Las operaciones se pueden implementar como métodos o como un evento de llamada que da lugar a una transición en la máquina de estados del objeto activo. Luego de invocada la operación, el emisor queda a la espera de que esta finalice para poder recuperar el control, recibiendo un valor de retorno si es que la operación lo produce.
Un elemento importante del diagrama de clases para esta tesis, es su metamodelo. En el metamodelo del diagrama de clases el paquete `Basic` de `InfrastructureLibrary::Core` [UMLInfra:2009] proporciona un lenguaje de modelado mínimo basados en clases. Este lenguaje puede ser utilizado para construir lenguajes más complejos y a su vez ser utilizado por MOF.
Son cuatro los diagramas utilizados para especificar las metaclases en `Basic`: `Types`, `Classes`, `DataTypes` y `Packages`. El diagrama `Types` define las metaclases abstractas que se utilizan para nombrar y tipar los elementos[3] de un modelo. El diagrama `DataTypes` define las metaclases que definen los tipos de datos. El diagrama `Packages` define los constructores básicos relacionados con los paquetes[4] y su contenido. El diagrama `Classes` define los constructores para el modelado basado en clases.

[3] Un elemento es cualquier componente del modelo

[4] Un paquete es un contenedor para tipos y otros paquetes

La Figura 2.7 muestra algunas de las metaclases de UML y las relaciones entre ellas. Una clase es un tipo que tiene como instancias objetos, y posee atributos y operaciones los cuales pueden o no ser heredados (véase sección 2.4.2.1). La clase `Type`, superclase de la clase `Class`, es una clase abstracta que representa la noción general del tipo de un elemento tipado y restringe el conjunto de valores que el elemento tipado puede tomar.

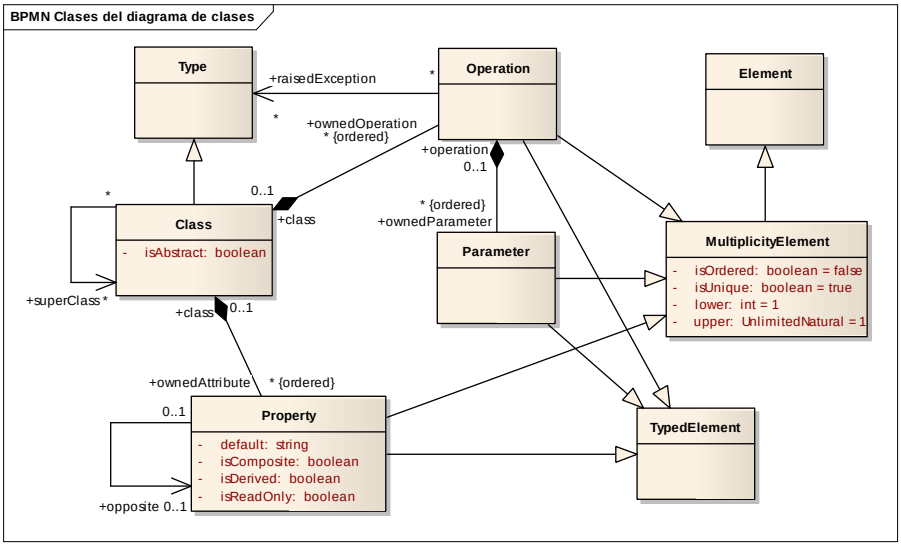

Figura 2.7: Las clases definidas en el metamodelo del diagrama de clases (adaptado de [UMLSuper:2009])

La metaclase `Class` tiene cuatro atributos: `isAbstract`, `superClass`, `ownedOperation` y `ownedAttribute`. El atributo `isAbstract` es de tipo booleano e indica si la clase es abstracta o no, esto es, si la clase puede o no tener instancias directas. Las clases abstractas se utilizan comúnmente para definir superclases. El atributo `ownedAttribute` indica los atributos poseidos por una clase, los cuales son instancias de la metaclase `Property`, y no incluye los atributos heredados. El atributo `ownedOperation` indica las operaciones poseidas por una clase, las cuales son instancias de la metaclase `Operation`, y no incluye las operaciones heredadas. El atributo `superClass` indica la superclase inmediata de la cual la clase hereda sus atributos y operaciones. Las metaclases `Property` y `Operation` tienen una relación de composición con la metaclase `Class`, esto quiere decir que existe una asociación con fuerte sentido de posesión y tiempo de vida coincidentes de las partes con el conjunto. De esta forma, las operaciones y las propiedades, viven y mueren con la clase, no tienen existencia propia por fuera de la clase.
Las metaclases `Property` y `Operation`, ambas heredan de las metaclases abstractas llamadas `MultiplicityElement` y `TypedElement`. La metaclase abstracta `MultiplicityElement` permite establecer la cardinalidad válida para una instancia de elementos tipados tales como extremos de asociación, atributos u operaciones. De esta forma, es posible que una clase tenga 0 o más, posiblemente infinitos, atributos u operaciones. También es posible establecer una multiplicidad

ordenada o única, cuando esta es multivaluada[5]. La metaclase `TypedElement` le da a las propiedades y operaciones un nombre y un tipo. El tipo de una operación, es el tipo del valor retornado por la operación.
Una operación pertenece a una clase, y cualquier objeto que sea una instancia directa o indirecta de la clase puede invocar la operación. El tipo de la operación, si lo hay, es el tipo del resultado retornado por la operación, y la multiplicidad es la multiplicidad del resultado. La operación puede asociarse con un conjunto de tipos, los cuales representan posibles excepciones que pueden dispararse al ejecutarse dicha operación.
De esta forma, es posible establecer que las operaciones pertenecen a una clase, no pueden existir sin la clase; las operaciones tienen parámetros, y estos no pueden existir, por la relación de composición que existe entre ambas metaclases, sin la operación; y existen un conjunto de tipos que relacionan las excepciones que se pueden producir al invocar una operación.
Una propiedad representa un atributo de una clase, tiene un tipo y una multiplicidad. Debido a la relación de composición que existe entre la clase y la propiedad, esta última no puede existir sin la clase. Una propiedad puede ser de sólo lectura, compuesta o derivada. La propiedad es de sólo lectura si no puede ser escrita después de inicializada; es compuesta, si el objeto actúa como contenedor para los valores del atributo; es derivada, si el valor del atributo se obtiene de información contenida en otro lugar. Igualmente, la propiedad puede tener un atributo de tipo string que es evaluado para darle un valor inicial cuando el objeto es instanciado, y puede tener un par de atributos emparejados, perteneciente al mismo objeto o a objetos diferentes, que jueguen el papel de opuestos, de tal forma que el atributo del objeto 1 se refiera al objeto 2 sí y sólo sí el atributo del objeto 2 se refiere al objeto 1.

Diagrama de componentes

Un componente es un artefacto que implementa un conjunto de interfaces. El hecho de implementar interfaces, permite que, en cualquier momento, el componente pueda ser reemplazado por otro. El diagrama de componentes muestra las partes internas, puertos y conectores que implementan un componente. Los puertos y conectores permiten las interacciones dentro y fuera del componente encapsulado.

Diagrama de estructura compuesta

Este diagrama muestra la estructura interna de una clase o una colaboración. La diferencia entre componentes y estructura compuesta es mínima. Por esta razón, este diagrama es poco utilizado.

Diagrama de objetos

Este diagrama permite modelar las instancias de los elementos existentes en el diagrama de clases, esto es, los objetos y sus relaciones en un momento dado. El diagrama de objetos permite expresar

[5] Una multiplicidad multivaluada es cuando el límite superior del intervalo es mayor que 1

la parte estática de una interacción, y consta de los objetos que colaboran, pero sin los mensajes que se intercambian entre ellos.
Este diagrama sirve para explorar relaciones entre objetos con el fin de validar el diagrama de clases o el código escrito, pero normalmente siempre hace parte de diagramas temporales del modelo.

Diagrama de despliegue

El elemento más importante que interviene en la construcción de este diagrama es el nodo. Este diagrama se utiliza al momento de modelar aspectos estáticos de nodos físicos y sus relaciones. Con este diagrama, es posible modelar muchos de los aspectos hardware de un sistema con el nivel de detalle adecuado para que se pueda especificar la plataforma sobre la que se ejecutará, de tal forma que la frontera software hardware del sistema sea mejor manejada.

Diagrama de paquetes

Con el fin de hacer mas simples los modelos y facilitar su compresión, se dispone de este diagrama. El diagrama de paquetes permite organizar los elementos de un modelo UML en grupos, esto es, pueden crearse paquetes de clases, paquetes de casos de uso, entre otros. No se recomienda que un paquete tenga ni muchos elementos, porque sería difícil de entender, ni muy pocos, porque no justificaría la agrupación en paquetes.
Los paquetes deben ser cohesivos, esto es, cualquier elemento que se agregue al paquete debe tener sentido cuando se considere el resto de elementos del paquete.
La utilidad de este diagrama es discutible, y muchos autores defienden el uso del diagrama de componentes en lugar del diagrama de paquetes.

2.4.2.2. Diagramas de comportamiento

El uso principal de estos diagramas es la visualización, especificación, construcción y documentación de los aspectos dinámicos del sistema. Los aspectos dinámicos de un sistema software incluyen flujo de mensajes a lo largo del tiempo y movimiento físico de componentes en una red.

Diagrama de casos de uso

Un caso de uso incluye un conjunto de secuencias de acciones, y sus variantes, que se ejecutan en el sistema con el fin de producir un resultado de valor para el actor. Los actores representan un conjunto coherente de roles que los usuarios de los casos de uso juegan al interactuar con estos.
El diagrama de casos de uso, permite modelar el comportamiento de un sistema, un subsistema o una clase. La mayoría de las veces, esto implica modelar el contexto.
Este diagrama se debe construir o por lo menos validar con los interesados. Debe tenerse en cuenta que el contenido es más importante que la representación, y por lo tanto es probable que el sólo

diagrama no sea suficiente para los propósitos que se buscan, lo que exigiría una especificación textual del mismo.

Diagrama de actividades

Este diagrama se asemeja a un diagrama de flujo donde se representa el flujo de control entre actividades. La actividad es una ejecución no atómica, producida por un conjunto de acciones individuales, en flujo secuencial o ramificado, cada una de las cuales puede producir un cambio en el estado del sistema o la comunicación de mensajes. En este diagrama, fácilmente se pueden visualizar las concurrencias y bifurcaciones de los flujos de control. Es bastante útil, y bien manejado puede reemplazar al diagrama de procesos utilizado en UN-METODO [Arango:2006].

Diagrama de estados

El comportamiento de un objeto se puede modelar mediante una máquina de estados, esto es, un comportamiento que especifica las secuencias de estados por las que pasa un objeto. Un estado indica una condición o situación en la vida de un objeto durante la cual satisface una condición, realiza una actividad o espera un evento.
En general, los diagramas de estado modelan la vida de las instancias de una clase, un caso de uso o un sistema completo. Estas instancias pueden responder a eventos tales como señales, operaciones o el paso del tiempo.
Sólo cuando la instancia del artefacto presenta diferentes y complejos comportamientos a lo largo de su vida, se justifica la utilización de este diagrama. Sin embargo, si el sistema es de tiempo real, el uso de este diagrama es generalizado.

Diagrama de secuencias

Un diagrama de secuencias permite visualizar la ordenación temporal de los mensajes. Se destaca, en este diagrama, los roles y los mensajes enviados y recibidos por las instancias de los roles.
El elemento principal del diagrama de secuencias son los mensajes. Los mensajes comunican información entre objetos, con la expectativa de que se desencadene una actividad. La llegada de un mensaje es básicamente la ocurrencia de un evento.
A pesar de lo útil que es este diagrama en secuencias complejas, no deben utilizarse de manera generalizada a lo largo de todo el sistema, ya que el aporte del diagrama a una secuencia simple es bastante pobre.

Diagrama de tiempo

Este diagrama permite visualizar las interacciones del sistema pero teniendo siempre en mente el tiempo. Permite visualizar el comportamiento de un clasificador o su interacción con otros clasifica-

dores, pero siempre enfocando su atención en el tiempo, esto es, instántes de ocurrencia de eventos que producen cambios en los clasificadores modelados.
En el diagrama de tiempo de UML se pueden definir los participantes de la interacción por medio de líneas de vida, a cada una de las cuales se pueden asociar múltiples estados, esto es, una condición de la línea de vida a la cual se llega o la cual se abandona cuando ocurre un evento.
El tiempo en este diagrama se representa en el eje horizontal en escala lineal, y puede mostrarse como una medida exacta o relativa. La medida relativa es muy útil, ya que en múltiples situaciones no se conoce cuando sucederá el evento que disparará el cambio de estado.
Para completar la información del diagrama de tiempo, es necesario adicionar restricciones temporales. Esto permite especificar, por ejemplo, el tiempo que transcurre mientras un participante se encuentra en un estado determinado, o el instánte en el que se produce un evento. Existen varios formatos para representar estas restricciones temporales.
Por sus características, la utilización de este tipo de diagrama está cogiendo fuerza en el modelado de sistemas embebidos de tiempo real.
El metamodelo del diagrama de tiempo de UML puede verse en la Figura 2.8 [UMLSuper:2009] [Muneton:2008]. Este metamodelo permite la construcción de diagramas de tiempo con líneas de vida, estados, y llamada a operaciones en los mensajes. `Interaction`, agrupa las líneas de vida y los mensajes de la interacción modelada. La relación entre las metaclases `Lifeline`, `Property`, `Message` y `OcurrenceSpecification`, permite modelar líneas de vida y cómo estas intercambian mensajes. Un mensaje define la comunicación en una interacción, la cual puede ser una señal, la llamada a una operación o la creación o destrucción de una instancia; igualmente, es posible determinar con el mensaje quién lo envía y quién lo recibe. Normalmente los mensajes están asociados a una `OcurrenceSpecification`, lo que permite identificar el mensaje enviado y su respuesta.
Los estados a los que puede llegar una línea de vida cuando se da un evento, pueden ser modelados por medio de la metaclase `StateInvariant`.
La metaclase `Property`, también presente en el metamodelo del diagrama de clases (véase Figura 2.7), representa el objeto contenido en la línea de vida del diagrama de tiempo. La metaclase `Operation`, también presente en el metamodelo del diagrama de clases (véase Figura 2.7), representa las operaciones invocadas por los mensajes, cuyos argumentos se asocian a los mensajes a través de la metaclase `Expression`.
El hecho de que el metamodelo del diagrama de tiempo y el metamodelo del diagrama de clases compartan algunas metaclases, por ejemplo `Operation` y `Property`, permite realizar trazabilidad al modelo y evaluar la consistencia del mismo, ya que, por ejemplo, la operación invocada por un mensaje debe estar asociadas a una clase en particular, lo que debe reflejarse en el diagrama de clases del modelo.

La especificación de la ejecución de una unidad de comportamiento o de una acción dentro de una línea de vida es realizada por `ExecutionSpecification`. La duración de esta unidad de ejecución se representa por dos `ExecutionOcurrenceSpecification`, etiquetadas como `start` y `finish` de acuerdo a la relación de herencia con la metaclase `OcurrenceSpecification`. Como la duración de la unidad de ejecución debe ser un valor positivo, entonces las metaclases `ExecutionOcurrenceSpecification` que definen el punto inicial y final de esta, deben ej-

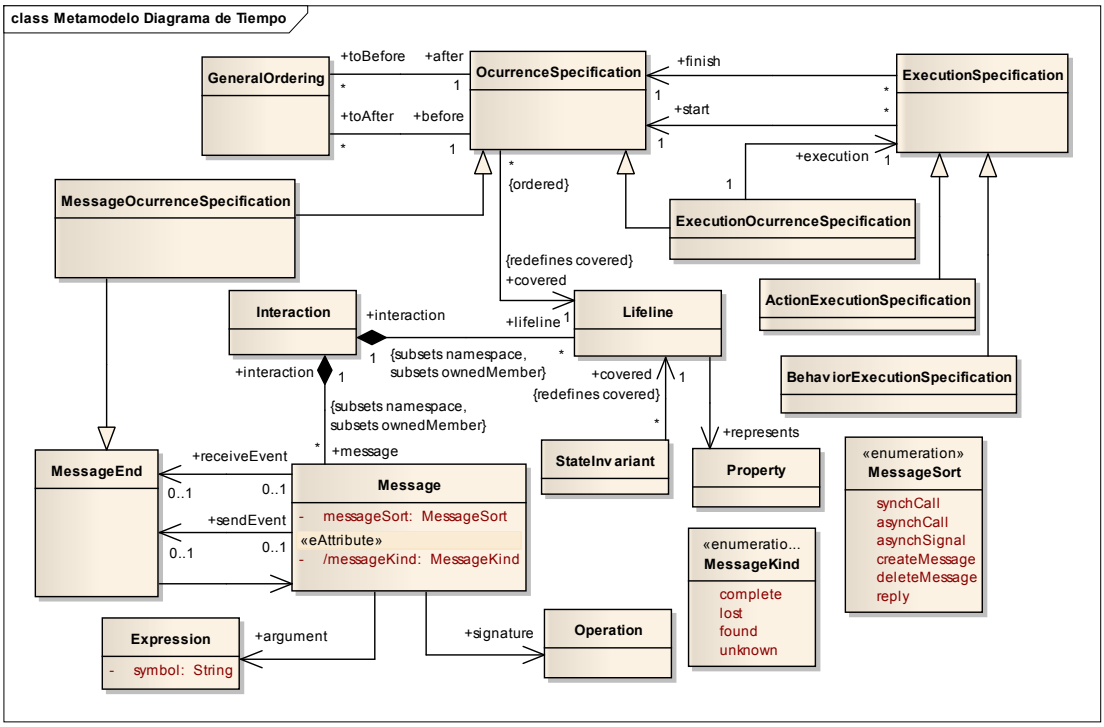

Figura 2.8: Metamodelo resumido del diagrama de tiempo (adaptado de [UMLSuper:2009] y [Muneton:2008])

cutarse de manera ordenada a lo largo de la línea de vida, lo cual se hace por medio de la metaclase `GeneralOrdering`.

Diagrama global de interacción

Este diagrama es una variante del diagrama de actividades. En este diagrama se representa el control de flujo dentro del sistema o proceso de negocio. Los nodos aquí son marcos, que pueden incluir otros diagramas, en lugar de actividades. El diagrama global de interacción es poco utilizado, debido en parte a la falta de espacio, y al hecho de la existencia del diagrama de actividades.

Diagrama de comunicación

Este diagrama permite establecer la organización de los objetos que participan en una interacción. La representación que se utiliza en estos diagramas, facilita visualizar claramente el flujo de control en el contexto de la organización estructural de los objetos que colaboran. Al ser semánticamente equivalente al diagrama de secuencias [UMLSuper:2009], se puede partir de un diagrama en una forma y convertirlo a la otra sin pérdida de información. Esto no significa que ambos diagramas visualicen explicitamente la misma información.

Si se trabaja con el diagrama de casos de uso y el diagrama de secuencias, en general resulta poco útil este diagrama de comunicaciones para modelar los aspectos comportamentales. Si el interés del modelador es por los aspectos estructurales del sistema, este diagrama puede resultar bastante útil.

2.5. Perfiles UML

Los perfiles son mecanismos que han sido especificamente definidos para extender UML a dominios especificos. Esta extensión debe garantizar la consistencia con el lenguaje y ser sintácticamente correcta. Los perfiles se apoyan en estereotipos y valores etiquetados, los cuales pueden aplicarse a, prácticamente, cualquier elemento del modelo.

2.5.1. Perfil para modelado y análisis de sistemas embebidos de tiempo real

El perfil UML para análisis y modelado de sistemas embebidos de tiempo real (MARTE – Modeling and Analysis of Real-Time Embedded Systems) [MARTE:2009], realiza una extensión de UML a dos dominios específicos: sistemas embebidos y tiempo real.
Un sistema embebido tiene varias características que sirven para su definición, pero en nuestro caso una muy importante es que sus componentes software no se exponen activamente al usuario. Además, se caracterizan por recursos de procesamiento, memoria y periféricos limitados, no proporcionan servicios computacionales estándar y usualmente forman parte de un sistema mayor, esto los hace dispositivos dedicados. Cuando se desarrolla software para esta clase de sistemas, esto es clave.
En la mayoría de las aplicaciones de la vida diaria, los sistemas de tiempo real a menudo trabajan en sistemas embebidos y por lo general los sistemas embebidos tienen necesidades de procesamiento de tiempo real.
En un sistema de tiempo real la puntualidad es el aspecto más importante. Tener que procesar múltiples entradas que llegan de forma impredecible con el fin de obtener las salidas correctas en el instante de tiempo apropiado, hace que se considere una falla del sistema, no sólo cuando la salida no es la correcta sino también cuando las restricciones de tiempo no se cumplen. Por lo tanto, los sistemas de tiempo real deben asegurar que las tareas las realiza correctamente y a tiempo.
Existen tres tipos de sistemas de acuerdo a las restricciones de tiempo: sistemas duros, sistemas suaves y sistemas firmes.
En los sistemas duros, una sola respuesta que no cumpla con las restricciones de tiempo se considera una falla del sistema. En los sistemas suaves, la puntualidad se especifica con un promedio de tiempo de respuesta; una respuesta que no cumpla con las restricciones de tiempo, usualmente no es significativa. Los sistemas firmes, son una combinación de sistemas duros y suaves, pero más cercanos a los requisitos de sistemas duros.
Los sistemas de tiempo real complejos y de gran tamaño demandan un gran esfuerzo de desarrollo inicial, ya que su arquitectura debe ser lo suficientemente flexible para adecuarse a los nuevos requisitos que a cada momento son identificados.

Los sistemas embebidos de tiempo real, son fundamentalmente sistemas donde la interacción hardware-software es muy estrecha, pero al igual que en todos los sistemas la funcionalidad debe ser especificada independientemente de la implementación. Se deben escoger plataformas flexibles con los suficientes servicios y recursos de tal forma que la funcionalidad pueda ser implementada.

2.5.1.1. Caraterísticas de MARTE

Las características con las cuales MARTE fue especificado son muy variadas, y permiten que los diferentes actores, usuarios de la especificación[6], cuenten con las siguientes posibilidades:

- Modelar, en forma independiente, tanto el software como el hardware del sistema RT/E
- Caracterizar los sistemas en forma cualitativa (comunicación, sincronización, paralelismos, entre otros) o cuantitativa (plazos, periodicidad, entre otros)
- Especificar o implementar los sistemas, partiendo de constructores de alto nivel y bajo nivel respectivamente
- Soportar las tendencias actuales en tecnologías empleadas en sistemas de tiempo real, paradigmas de diseño y técnicas de análisis de modelos. Pero igualmente, está abierto a nuevos desarrollos en esta dinámica área del conocimiento
- Realizar análisis de las características software y hardware del sistema RT/E en forma cuantitativa y segmentada

2.5.1.2. Enfoque y estructura de MARTE

El perfil se estructura alrededor de dos intereses principales: modelar las características de los sistemas RT/E y acotar los modelos de aplicación, de tal forma que se soporte el análisis de las propiedades del sistema (véase Figura 2.9). El modelado está soportado por el paquete “MARTE design model”, mientras que el análisis depende del paquete “MARTE analysis model”. Ambos paquetes están centrados en el tiempo y la utilización de recursos concurrentes, los cuales están contenidos en el paquete compartido “MARTE foundations”. Finalmente, la característica de análisis del modelo consta de dos partes: un paquete genérico GQAM, y dos paquetes específicos para dominios que tienen que ver enteramente con tiempo. El hecho de que los dominios específicos estén relacionados directamente con tiempo no es una limitante, ya que es posible definir dominios de análisis adicionales, tales como consumo de potencia, uso de la memoria, confiabilidad, entre otros.

2.5.1.3. Cimientos de MARTE

MARTE está fundamentado en cinco aspectos: los elementos definidos en el core – CoreElements, el modelado de propiedades no funcionales – NFP, el modelado del tiempo – Time, el modelado de recursos genéricos – GRM y el modelado de asignación de recursos – Alloc.

[6] Arquitectos del sistema RT/E, modeladores y arquitectos hardware, modeladores y arquitectos software, analistas del modelo, proveedores de la plataforma de ejecución, proveedores de la metodología de diseño y análisis, entre otros

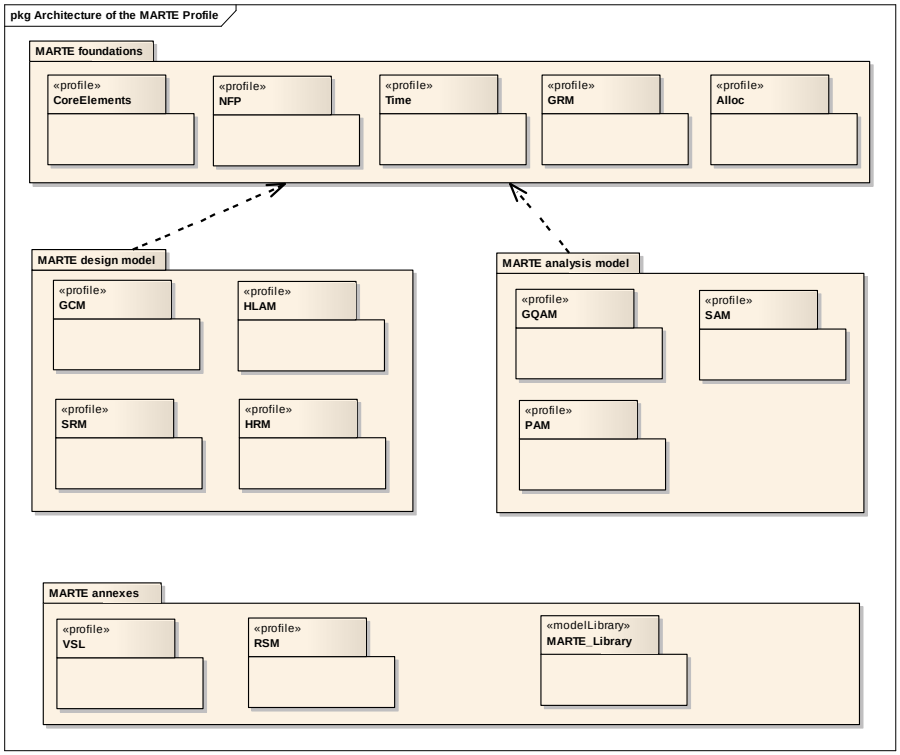

Figura 2.9: Arquitectura del perfil MARTE [MARTE:2009]

Elementos del Core

Los elementos definidos aquí, `CoreElements`, son la base para el resto de la especificación. Este conjunto completo de conceptos relacionados, es útil para definir otros más elaborados, y todo está basado en dos paquetes centrales: `Foundations` y `Causality`.
El paquete `Foundations`, incluye los elementos básicos utilizados para el modelamiento estructural, e igualmente sus conceptos pueden ser utilizados para propósitos de análisis. Para el análisis y modelado, es fundamental distinguir entre clasificadores en tiempo de diseño (clases, tipos, entre otros), e instancias en tiempo de ejecución, las cuales son creadas sobre la base de los clasificadores. Un clasificador dado puede tener cualquier cantidad de instancias, y una instancia puede tener múltiples tipos (léase clasificadores) (véase Figura 2.10).

A los clasificadores, como a cualquier elemento del modelo, se les puede asociar valores de NFPs, lo que significa que dichos valores se aplican a todas las instancias del clasificador. Sin embargo, estos valores pueden ser sobreescritos por valores redefinidos en cada instancia del clasificador.
Todos los elementos del modelo en cualquier nivel de especificación, se pueden representar por uno de los elementos fundamentales del paquete `Foundations`, clasificador o instancia.
El paquete `Causality`, describe los elementos necesarios para modelar comportamiento y comprender la semántica en tiempo de ejecución de aquellos elementos del modelo que son utiliza-

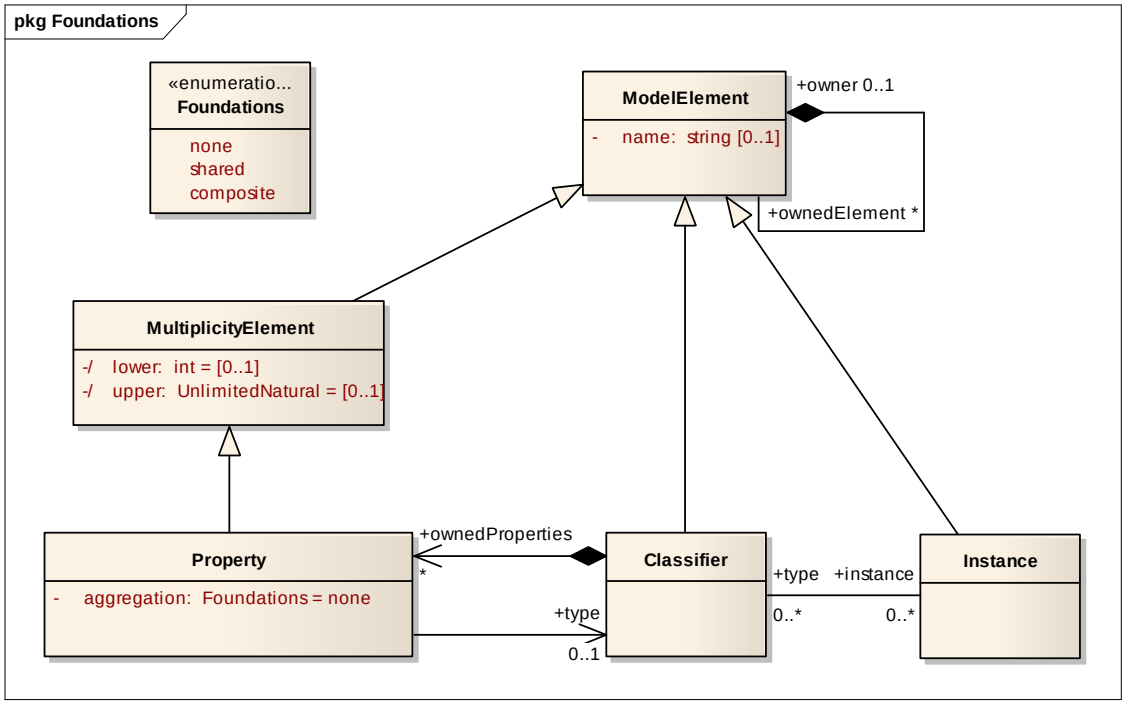

Figura 2.10: Diagrama de clasificadores, instancias y propiedades del paquete Foundations [MARTE:2009]

dos en sistemas RT/E. Este paquete está compuesto por cuatro paquetes: `CommonBehavior`, `Invocation`, `RunTimeContext` y `Communication` (véase Figura 2.11).

El comportamiento describe como un sistema o una entidad cambian en el tiempo. Este comportamiento puede tener parámetros asociados, cuyos valores podrian utilizarse para evaluar el comportamiento en sí. La especificación de un comportamiento, el cual se utiliza para representar el comportamiento que se ejecuta al momento de crearse una instancia de un clasificador, se denomina comportamiento principal. Por ejemplo, en sistemas de tiempo real concurrentes, el comportamiento principal puede ser el comportamiento que inicia la actividad de un hilo, el cual continua hasta que el hilo finaliza.

Dentro del paquete `CommonBehavior`, dos tipos de comportamiento pueden definirse, estos son: `CompositeBehavior` y `Action`. `Action` es un comportamiento atómico, mientras que `CompositeBehavior` puede contener otros comportamientos, los cuales pueden ser compuestos o atómicos. Junto con `Action`, `Event` y `Trigger`, son tres los elementos fundamentales que se utilizan para representar comportamiento de un sistema. `Action` es la unidad fundamental de comportamiento; `Event` es la especificación de un tipo de cambio de estado que puede suceder en un sistema modelado, la ocurrencia de un `Event` es a menudo el producto de algunas acciones dentro del sistema o en su entorno; `Trigger` puede desatar la ejecución de un comportamiento, esto es, el `Trigger` es el concepto que relaciona un evento al comportamiento que puede afectar cualquier instancia del clasificador asociado.

`BehaviorExecution` es una especificación de la ejecución de una unidad de comportamiento

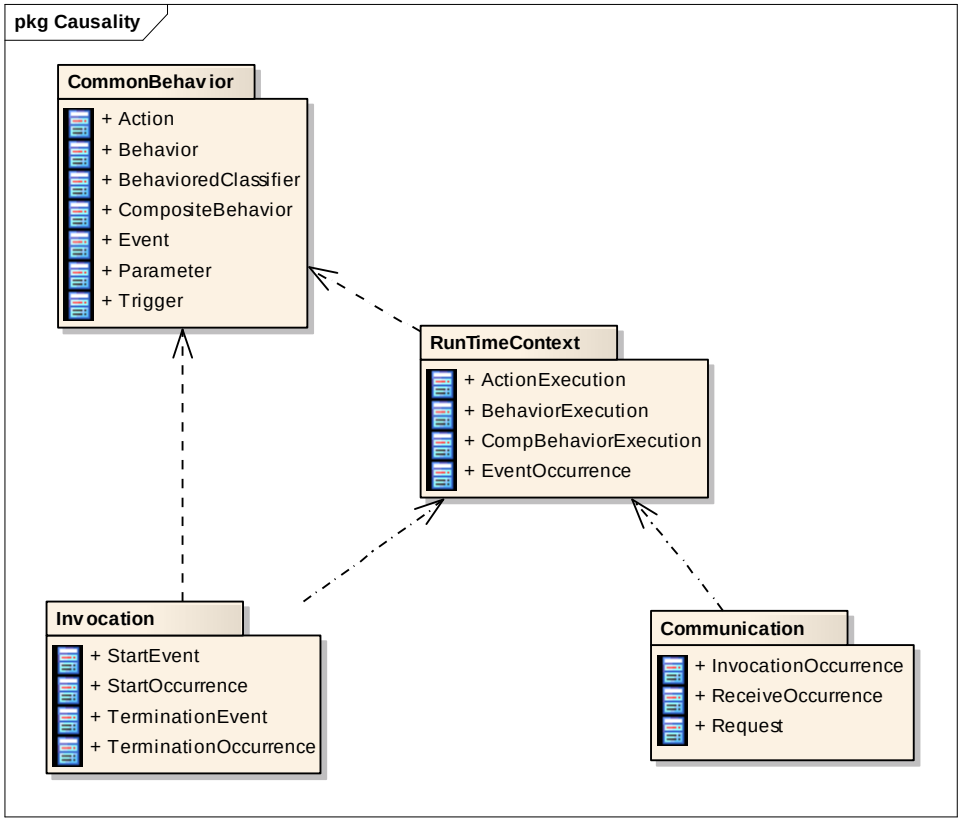

Figura 2.11: Arquitectura del paquete Causality [MARTE:2009]

dentro de la instancia de un `BehavioredClassifiers`. Por lo tanto, la ejecución de un comportamiento son instancias en tiempo de ejecución de los conceptos comportamiento y accción, los cuales se instancian con `CompBehaviorExecution` y `ActionExecution`. Igual sucede para los eventos, `EventOccurrences`. De esta forma, cualquier ejecución de comportamiento es la consecuencia directa de la ejecución de una acción de por lo menos una instancia de un clasificador.
El paquete `Invocation` es útil cuando se desea considerar el inicio y la finalización de un comportamiento, de esta manera marcas de tiempo pueden ser adicionadas al diagrama con el fin de realizar un posterior análisis del modelo.
El paquete `Communication` permite la comunicación entre instancias de clasificadores y el llamado a unidades de comportamiento. Las instancias responden a mensajes generados por la ejecución de acciones de comunicación. Cuando estos mensajes llegan, el receptor responde ejecutando el comportamiento que coincide con la solicitud hecha por el mensaje.
Una acción que representa el llamado a una característica comportamental, es ejecutada por una instancia de un clasificador, denominada instancia fuente, resultando en un `InvocationOccurrence`. El evento invocado puede representar el envío de una señal o la llamada a una operación, generándose un requisito. Un requisito, el cual corresponde completamente al concepto `Request` de UML [UMLSuper:2009], es una instancia de una comunicación en tránsito.

Modelado de propiedades no funcionales

El subpaquete NFPs de la especificación MARTE [MARTE:2009] provee un marco de trabajo general para agregar notas de NFPs a elementos de modelos UML. NFPs se centra en la formalización de un conjunto de constructores de modelado que permiten especificar esta clase de propiedades de una manera detallada. Este marco de trabajo se enfoca en:

- Cómo y cuáles NFPs describir
- Cómo se deben agregar instancias particulares de NFPs a elementos de modelos UML
- Cómo definir relaciones entre diferentes NFPs
- Cómo expresar restricciones sobre NFPs o entre ellos
- Minimizar el esfuerzo del diseñador al agregar notas de NFPs a elementos de modelos UML
- Ofrecer un marco de trabajo abierto para especificar NFPs sobre elementos UML de cualquier dominio de interés

Cuando modelamos sistemas de cómputo, nos ocupamos de su arquitectura y comportamiento utilizando elementos del modelo (como recursos, servicios ofrecidos por los recursos, características de comportamiento, operaciones lógicas, modos de configuración, vistas de modelos, entre otros) y sus propiedades. Como en el caso de los requisitos (véase sección 2.2) las propiedades no funcionales se agrupan en dos grandes categorias: propiedades funcionales y propiedades no funcionales. Las primeras se ocupan del propósito de la aplicación, las segundas de como la aplicación alcanza su propósito.
En el mundo de los sistemas RT/E modelar NFPs es fundamental para el éxito del sistema al momento de su implementación, ya que características como throughput, retardo, politicas de planificación, cumplimiento de plazos o utilización de memoria, son necesariamente decisiones de diseño. la estructura general del paquete que soporta el marco de trabajo para las NFPs puede verse en la Figura 2.12.

Un NFP puede ser cuantitativo o cualitativo. `QuantitativeNFP` representa una propiedad medible, ya sea por medio de un conjunto de valores o una función que la caracterice. Las medidas son una cantidad representadas por una unidad específica, la cual puede ser básica o derivada.
Las propiedades cualitativas, `QualitativeNFP`, se refieren a características distintivas que no pueden ser medibles directamente. En general, se representa por una etiqueta que indica un nivel de abstracción muy alto, útil para el análisis del sistema. De otra forma podemos indicar, las NFP cualitativas toman un valor de una lista de valores permitidos, donde cada valor identifica una posible alternativa.
El paquete `NFP_Annotation`, permite enriquecer la semántica de los modelos adicionando notas a los elementos del modelo UML. Estas notas se refieren a NFPs, las cuales son muy útiles ya que pueden contextualizar el modelo a dominios específicos, permitiendo extraer de allí su ontología, y preparando el modelo para diferentes tipos de análisis, por ejemplo de desempeño o planificación.
El paquete `NFP_Declaration`, está destinado para calificar y asignar tipos de datos extendidos a valores NFP. Aquí se introduce la noción de valor de propiedad, `ValueProperty`, la cual se especializa en el concepto de NFP. Un valor de propiedad representa cualquier clase de cantidad física relevante en el diseño del sistema. Un NFP es una clase de valor de propiedad, enfocada en los aspectos que hacen parte del contexto del sistema.

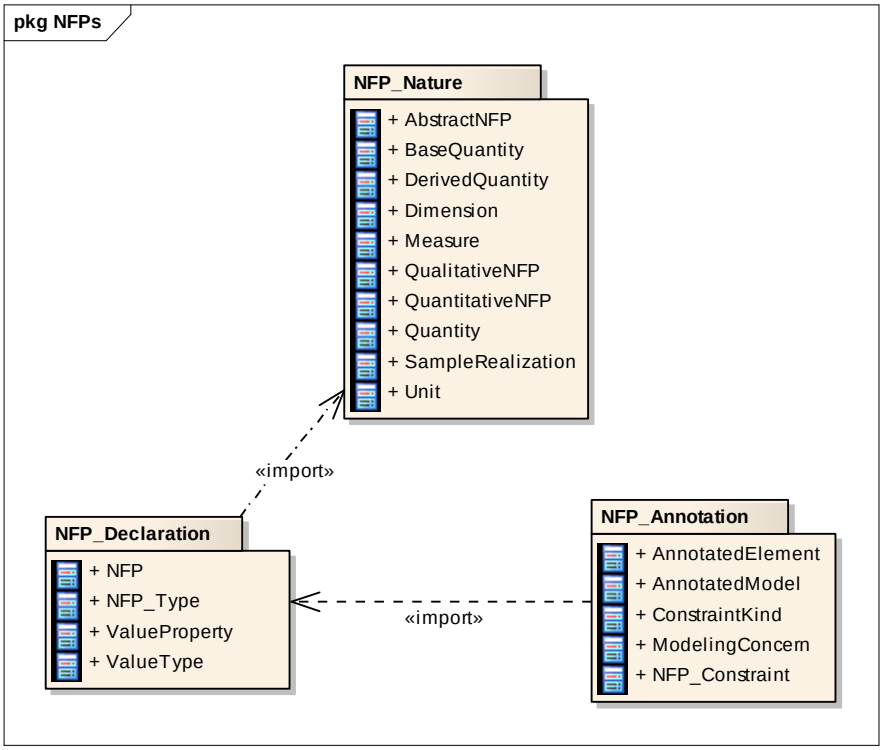

Figura 2.12: Estructura y dependencias del paquete NFPs [MARTE:2009]

Un `ValueType` adiciona la habilidad de ponerle una unidad de medida y un cualificador adicional a los valores. Un `ValueType` con una unidad de medida se asocia a una cantidad física. Los clasificadores `NFP` y `NFP_Type` son especializaciones directas de `ValueProperty` y `ValueType` utilizados para la descripción de aspectos no funcionales de un sistema.

Modelado del Tiempo

MARTE define un marco de trabajo general para representar tiempo y los conceptos relacionados con este, así como los mecanismos que son apropiados para modelar sistemas RT/E. La cardinalidad del tiempo es imprescindible para los sistemas de tiempo real, y en algunos casos el tiempo lógico es necesario para el modelado de los sistemas embebidos. Debido a esto, las definiciones dadas en MARTE cubren métricas de tiemepo y tiempo lógico.
Este modelo es bastante general, y es altamente improbable que para un sistema en particular se requiera todo el conjunto de sus conceptos y semántica. Sin embargo, en las diferentes fases del desarrollo de un sistema RT/E (modelado, diseño, análisis de desempeño, análisis de planificación, implementación) el tiempo puede percibirse diferente, y esto hace que se requiera la existencia de tres abstracciones diferentes del concepto tiempo que permiten representar flujos de comportamiento, a saber: causal/temporal, basado en reloj/síncrono y físico/tiempo real.
El modelo causal/temporal se interesa exclusivamente por la relación precedencia/dependencia de

las instrucciones, relación que puede ser parcial cuando se trata de sistemas concurrentes.
El modelo basado en reloj/síncrono adiciona la noción de simultaneidad, y divide la escala de tiempo en una sucesión discreta de instántes. Es común en este modelo que la unidad de medida sea el tick del reloj, ya que el reloj se asocia con un tren de pulsos periódico.
El modelo físico/tiempo real permite el modelado de valores de tiempo muy precisos, los cuales se utilizan para propósitos de planificación en sistemas críticos.
El comportamiento de un sistema y el tiempo son conceptos fuertemente acoplados, y por esto, el tiempo no debe considerarse como un modelo externo cuando se trata de modelar sistemas RT/E. Por ejemplo, el paquete `Causality` proporciona una vista de alto nivel de la semántica en tiempo de ejecución de sistemas RT/E, mientras que el paquete `Invocation` es extendido con el concepto de `SimultaneousOccurrenceSet`.
La estructura general del modelo del dominio del tiempo puede verse en la Figura 2.13; los paquetes `BasicTimeModels` y `MultipleTimeModels` proveen el modelo estructural del tiempo que constituye los cimientos semánticos del enfoque. Estos dos paquetes se fusionan, ya que el concepto de base de tiempo, `TimeBase`, introducido en el paquete `BasicTimeModels` es enriquecido posteriormente en el paquete `MultipleTimeModels`. Ambos paquetes son utilizados por `TimeAccesses` y `TimeRelatedEntities`, ya que estos últimos contienen conceptos y constructores utilizados por el usuario estándar del perfil.

El paquete `BasicTimeModels` provee una vista estructural del tiempo como un conjunto ordenado de instántes, aquí no se hace referencia a tiempo físico, ya que ignora la duración entre instantes sucesivos. Debido a que el modelo se enfoca exclusivamente en la ordenación de instantes, soporta adecuadamente el tiempo lógico.
El modelo de tiempo básico de MARTE [MARTE:2009] posee un contener ordenado de instántes, `TimeBase`, en el cual se puede definir la estructura del tiempo, denso o discreto (véase Figura 2.14). Definir el tiempo como denso significa que para cualquier par de instantes, siempre existirá un instánte entre ellos. El conjunto de instántes es numerable, esto es, los instántes pueden ser indexados por números enteros positivos si el tiempo es discreto, mientras que para tiempo denso la indexación se hace con números racionales. Un modelo de tiempo continuo requiere indices reales, lo que haría al conjunto de instántes no numerable, y por lo tanto no es soportado por esta especificación. En la práctica, la naturaleza contable del conjunto de instántes, no es una limitación para el modelado de sistemas.

`DiscreteTimeBase` es una subclase de `TimeBase`, y se definió debido a que la base de tiempo discreto juega un rol central en el modelo estructurado del tiempo. `JunctionInstant` son instántes especializados, y una base de tiempo discreta sólo posee `JunctionInstant`, esto no indica que una base de tiempo denso no pueda poseer sus propios `JunctionInstant`.
Existen algunos tipos de sistemas, como los distribuidos, para los cuales una visión lineal del tiempo, como la mostrada en la Figura 2.14, no es suficiente. En estos casos, es necesario tener múltiples bases de tiempo (véase Figura 2.15), esto es, la estructura del tiempo contiene un árbol de varias bases de tiempo relacionadas. Los instantes involucrados en tales relaciones son instantes del tipo `JunctionInstant`.
De la clase abstracta `TimeInstantRelation` se derivan tres subclases concretas. La primera es `CoincidenceRelation`, la cual permite modelar simultaneidad entre `JunctionInstant` pertenecientes a diferentes bases de tiempo. La segunda es `PrecedenceRelation`, la cual ex-

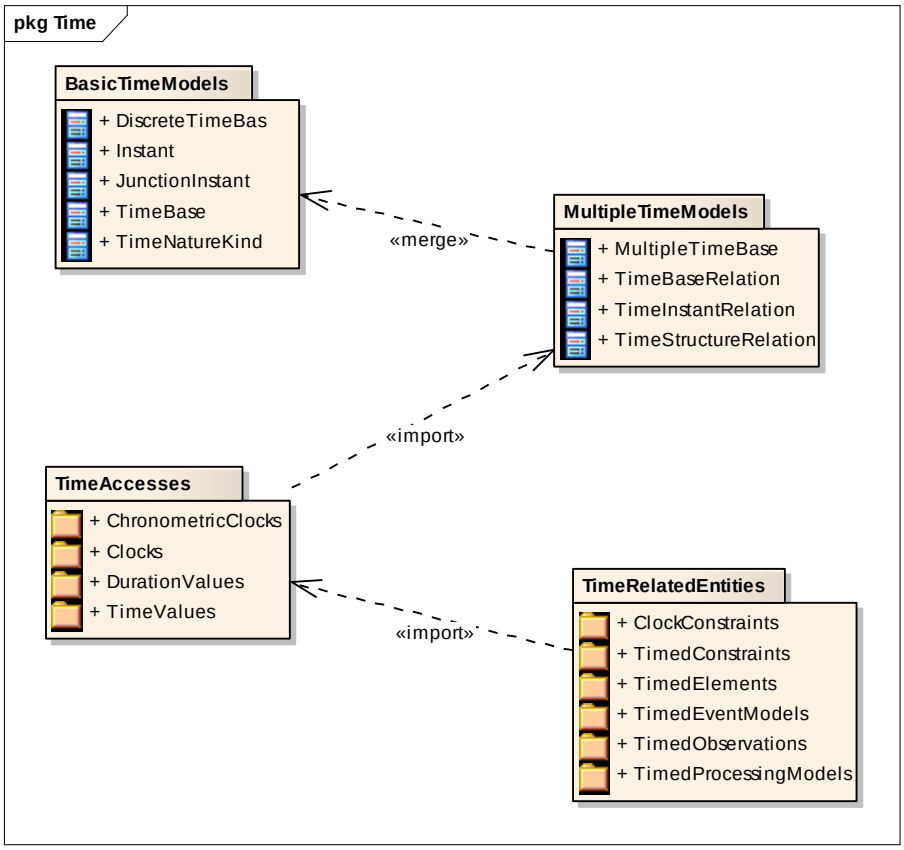

Figura 2.13: Estructura del modelo del dominio del tiempo [MARTE:2009]

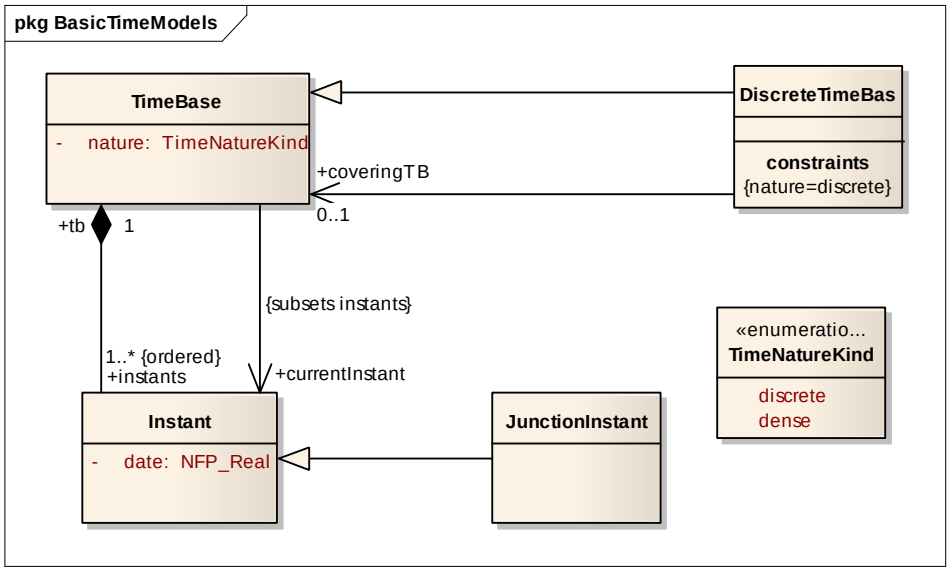

Figura 2.14: Modelo de tiempo básico [MARTE:2009]

presa una dependencia direccional, ya que un `JunctionInstant` perteneciente a una base de tiempo puede preceder o suceder a un `JunctionInstant` perteneciente a una base de tiempo diferente. Y por último la clase concreta `TimeIntervalMembership`, la cual permite establecer si un grupo de `JunctionInstant` son coincidentes o no.

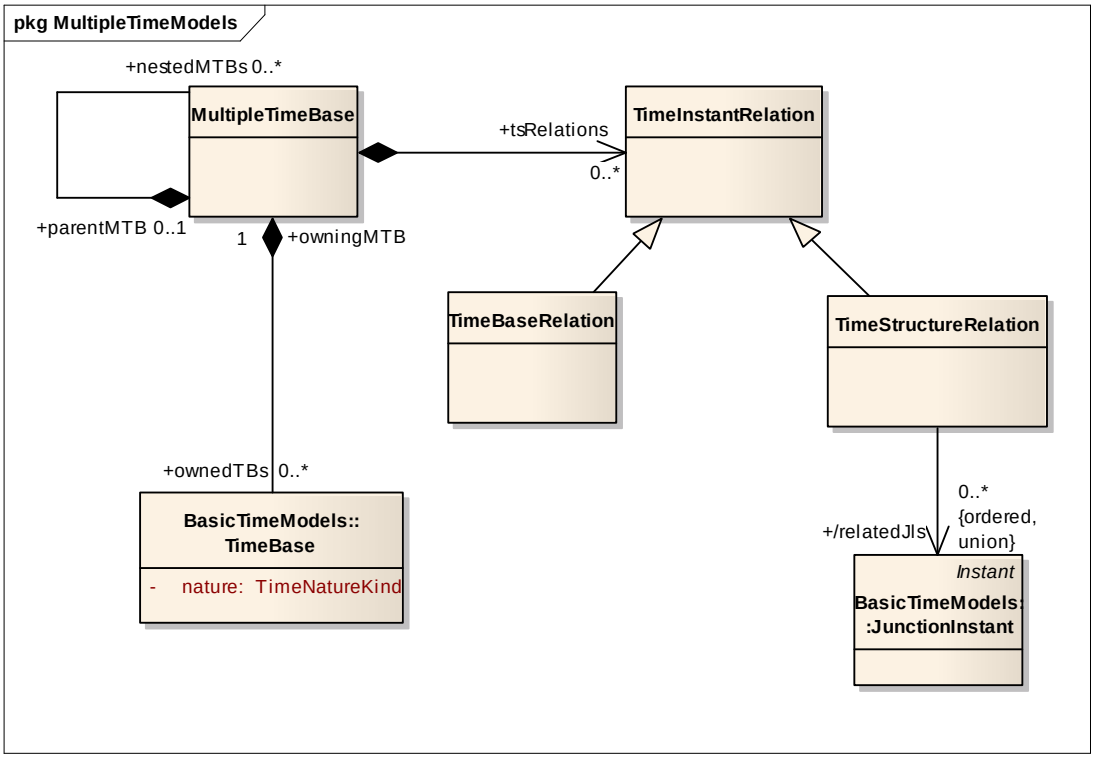

Figura 2.15: Modelo del tiempo con múltiples bases de tiempo [MARTE:2009]

Modelado de Recursos Genéricos

El recurso, es el elemento principal del GRM, y representa una entidad fisica o lógica que ofrece un servicio. El recurso, junto con el servicio que ofrece, ponen a disposición del sistema los medios necesarios para satisfacer los requisitos de este. Es posible asociar NFPs a los recursos.
Con el fin de modelar adecuadamente el sistema, es necesario diferenciar entre elementos de la aplicación y elementos de la plataforma. los elementos de la plataforma son considerados como recursos o servicios proporcionados por los recursos, esto es, la vista estructural se modela con recursos, mientras que la vista comportamental con servicios.
Los recursos, pueden clasificarse como pasivos o activos, de cómputo – `ComputingResource`, de comunicaciones – `CommunicationResource`, de almacenamiento – `StorageResource`, de temporización – `TimingResource`, de sincronización – `SynchResource`, de concurrencia – `ConcurrencyResource` o de dispositivos – `DeviceResource`, y protegidos o no prote-

gidos. Los recursos pasivos sólo pueden reaccionar a estímulos, los activos son capaces de generar sus propios estímulos sin que esto se les pida explícitamente. Los `ComputingResource` representan dispositivos de procesamiento físicos o virtuales, capaces de almacenar y ejecutar código de programa. Los `CommunicationResource` permiten la comunicación entre recursos. Los `StorageResource` representan memoria. Los `TimingResource` representan entidades software o hardware que están en capacidad de seguir y evidenciar el paso del tiempo. Los `SynchResource` facilitan el manejo de la concurrencia. Los `ConcurrencyResource` están en capacidad de establecer flujos de ejecución de manera concurrente con otros recursos. Los `DeviceResource` son otro tipo de recursos, que no representan recursos de los ya considerados. Los recursos protegidos o no protegidos, son aquellos cuyo acceso está restringido o no, de acuerdo a alguna política de control de acceso.

Modelado de Asignación de Recursos

Para los sistemas RT/E es de vital importancia la asignación de los elementos funcionales sobre los recursos disponibles, esto es, sobre la plataforma de ejecución. La asignación consiste en la planificación, tanto en el aspecto temporal como espacial, que debe tenerse en cuenta con el fin de mapear los algoritmos a ejecutar sobre los recursos y servicios de cómputo y comunicaciones. El perfil UML MARTE, incluye definiciones que son relevantes tanto para los modelos de la plataforma ejecución como para los modelos de aplicación. Cuando se establece una asociación entre la aplicación y la platafotrma de ejecución, se dice que se está realizando una asignación. Un elemento de aplicación será cualquier artefacto UML adecuado para modelar una aplicación, el cual tendrá aspectos estructurales y de comportamiento. Mientras que una plataforma de ejecución es representada por un conjunto interconectado de recursos, donde cada uno proporciona servicios que soportan la ejecución de la aplicación.
Los modelos de la plataforma de ejecución y aplicación se construyen por separado, para luego ser apareados por medio del proceso de asignación. Este proceso requiere un ajuste previo de cada modelo, con el fin de lograr una coincidencia directa entre los componentes del modelo. La asignación puede ser estructural, comportamental o hibrida. La primera es una asociación entre un grupo de elementos estructurales y un recurso. La comportamental es una asociación entre un conjunto de elementos comportamentales y un conjunto de servicios proporcionados por la plataforma de ejecución. Y cuando el contexto está claro, también puede representarse una asignación hibrida.

2.5.1.4. Modelo de Diseño de MARTE

Para modelar los sistemas RT/E se definieron tres elementos: los artefactos que permiten afrontar el modelado con un enfoque basado en componentes, esto es, el modelo de componentes genérico – GCM; igualmente, los conceptos de alto nivel para las aplicaciones; y finalmente, el modelado detallado de recursos software y hardware.

Modelado de Componentes Genéricos

El modelo del dominio introducido para GCM es principalmente una abstracción de las clases estructuradas de UML. Este modelo provee un denominador común entre varios componentes del modelo, los cuales en principio no apuntan exclusivamente al dominio de los sistemas RT/E. El propósito es el de darle a MARTE un modelo lo más general posible, que no esté atado a semánticas de ejecución específicas, sobre el que las características de tiempo real puedan ser posteriormente aplicadas.

Modelado de Aplicaciones de Alto Nivel

El paquete HLAM de MARTE proporciona conceptos de modelado de alto nivel, cuyo propósito es el de servir para modelar sistemas RT/E. A diferencia de dominios de aplicación usuales, los sistemas RT/E exigen posibilidades de modelado de características tanto cualitativas como cuantitativas.
La concurrencia es un aspecto clave cuando se están modelando sistemas RT/E. Con el fin de manejar esta característica, MARTE usa el concepto de `RtUnit` (véase Figura 2.16). Con `RtUnit` es posible crear elementos de alto nivel que permitan modelar sistemas RT/E, ya que son similares a los objetos activos de UML, pero con un nivel de detalle semántico mayor.
Cada `RtUnit` puede poseer un conjunto de recursos de planificación o crearlos dinámicamente cada vez que se requiera. De aquí, que una `RtUnit` puede verse como un recurso de ejecución autónomo, que puede manejar diferentes mensajes al mismo tiempo, por ejemplo, mensajes de concurrencia y de restricciones de tiempo real. Una `RtUnit` es una unidad de concurrencia que encapsula en una sola entidad los paradigmas de objeto y proceso. Cualquier unidad de tiempo real puede invocar servicios de otras unidades de tiempo real, enviando señales o datos, sin que esto perturbe las características de concurrencia del sistema.
Al modelar concurrencia, es mandatorio poder modelar información compartida, por lo tanto se ha introducido la `PpUnit` (véase Figura 2.16). La unidad pasiva protegida, `PpUnit`, especifica su política de concurrencia de manera global o local, mientras que el tipo de ejecución es `immediateRemote` o `deferred`, el cual en ambos casos es ejecución remota.

Modelado Detallado de los Recursos

Este apartado detalla un conjunto de recursos útiles para modelar tanto componentes software como hardware. El modelado de recursos software se enfoca en las interfaces de programación de aplicaciones de software multitarea. El modelado de recursos hardware se enfoca sobre plataformas hardware.
Los sistemas RT/E pueden ser construidos de una de estas dos formas: con un enfoque secuencial o con un enfoque multitarea. Con el primero se definen un conjunto de acciones ordenadas, las cuales están precalculadas para satisfacer los requisitos de tiempo real del sistema. Con el segundo enfoque, se considera que existen un conjunto de unidades que se ejecutan concurrentemente e interactuan entre ellas. MARTE utiliza este último enfoque, bajo un soporte de ejecución basado en un

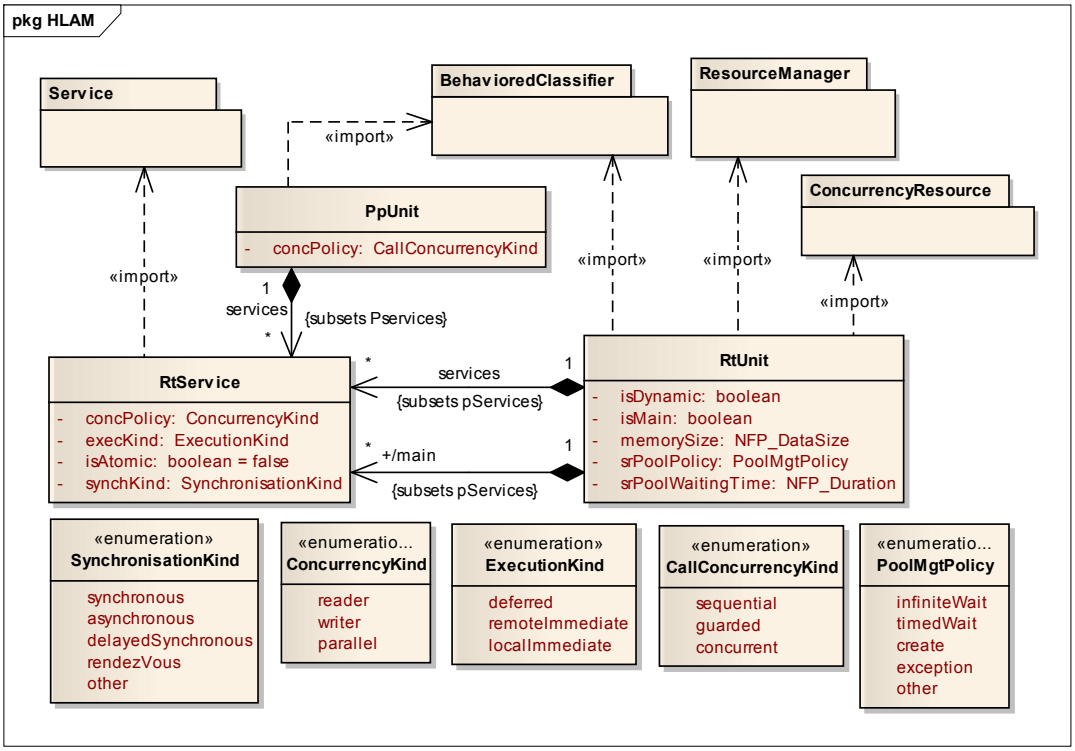

Figura 2.16: RtUnit y pPUnit del paquete HLAM [MARTE:2009]

RTOS. El uso típico del modelado de recursos software es la descripción de una manera unificada de APIs multitarea.
Regularmente los recursos software multitarea están relacionados con: contextos de ejecución concurrentes, interacción entre contextos concurrentes y administración de recuros software y hardware (véase Figura 2.17).
El paquete `SW_ResourceCore`, suministra al marco de trabajo los elementos necesarios para modelar tipos y operaciones para que las APIs incluyan los conceptos propios de los sistemas RT/E.
El paquete `SW_Concurrency`, define las entidades que compiten por los recursos de cómputo. Existen dos tipos de recursos concurrentes: los de interrupción y los de planificación. Los recursos de interrupción tienen que ver con las interrupciones hardware y las excepciones. Las excepciones pueden ser “detectadas por el procesador”, esto es, cuando la CPU detecta condiciones anómalas durante la ejecución de una instrucción, por ejemplo, una división por cero; o las excepciones pueden ser programadas, por ejemplo, una interrupción por software.

En contextos de ejecución concurrentes, los recursos interactuan con el fin de sincronizar sus acciones y comunicar los datos. La primera tarea la realiza `SwSynchronizationResources`, la segunda `SwCommunicationResources`. En cualquier caso, los recursos interactuan de acuerdo a una política de espera, los cuales se llevan a una cola caracterizada por su capacidad y la política de espera aplicada.

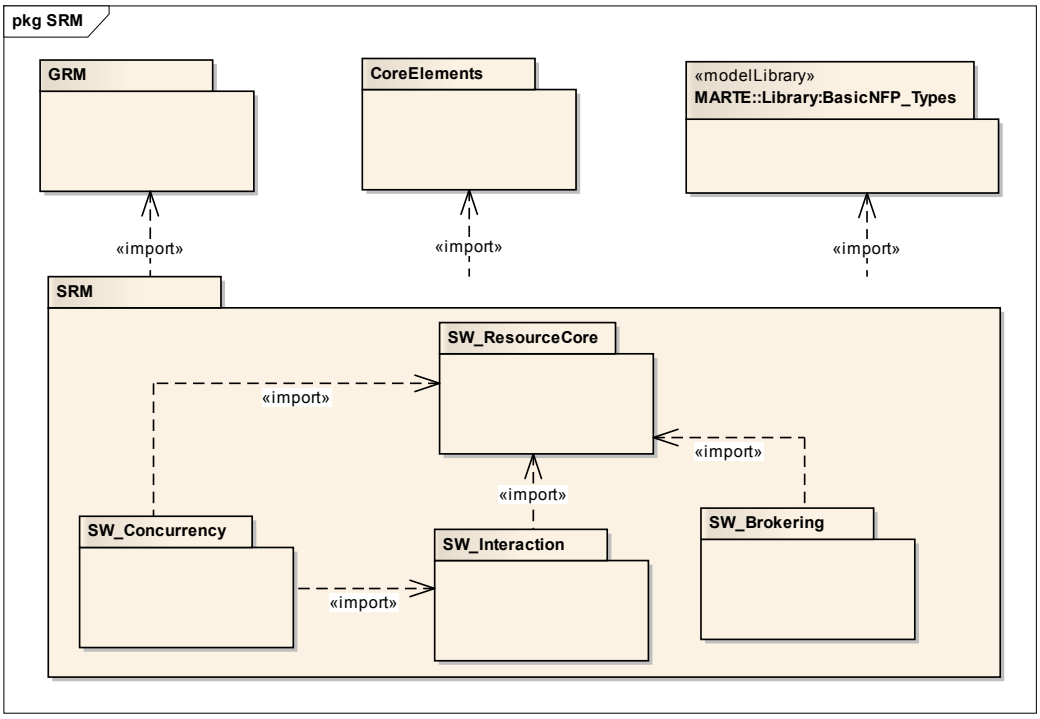

Figura 2.17: Estructura del marco de trabajo para modelar recursos software [MARTE:2009]

2.5.1.5. Modelo de Análisis de MARTE

Existen múltiples dominios de análisis sobre los cuales caracterizar el comportamiento del software, y cada uno tiene diferencias en su terminología, conceptos y semántica. Sin embargo, también comparten conceptos fundamentales de modelado básico y NFPs.
MARTE incluye dominios especializados para desempeño y planificación, pero su modelo de análisis es tan flexible que permite definir otros como consumo de energía, uso de la memoria, confiabilidad, disponibilidad, y seguridad, entre otros.
Con MARTE es posible realizar evaluaciones precisas y fidedignas de modelos de sistemas RT/E, utilizando análisis cuantitativo formal basado en modelos matemáticos. Esto le permite al diseñador detectar problemas en etapas previas del ciclo de vida del desarrollo, reduciendo los costos y el riesgo asociado al proyecto.
La Figura 2.18 muestra la relación entre el paquete GQAM y los dos sub-perfiles para planificación (SAM – 16Schedulability Analysis Modeling) y desempeño (PAM – Performance Analysis Modeling).

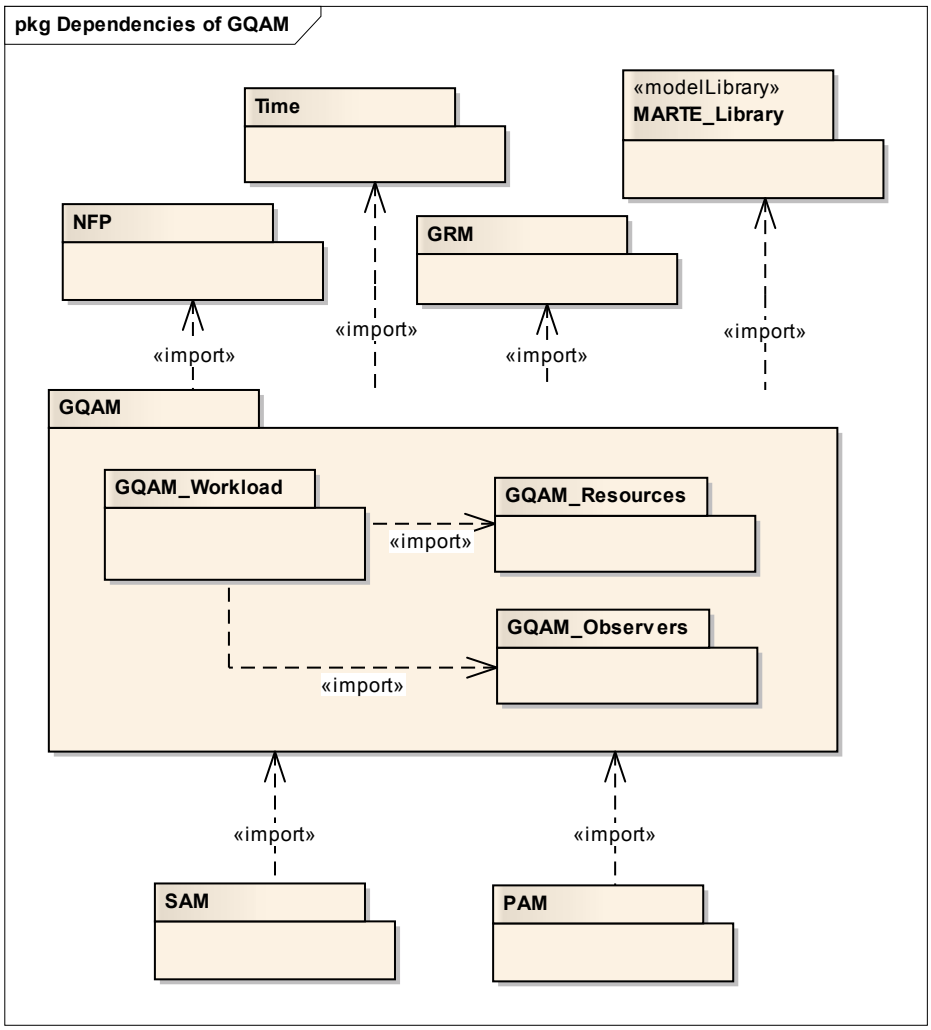

Figura 2.18: Dependencias del paquete GQAM [MARTE:2009]

Modelado de Análisis Cuantitativo Genérico

Para garantizar el desempeño de un sistema software, el proceso de desarrollo debe capturar los requisitos de desempeño en el contexto de diseño, asociar las características de desempeño relacionadas con QoS con elementos de un modelo, especificar los parámetros de ejecución que puede utilizar una herramienta de modelado para calcular las características de desempeño previstas y realizar las pruebas para determinar el desempeño del sistema [SPEC:2010] [IEEE1012:2005].
La especificación del desempeño deseado de un sistema, debe permitir estimar el desempeño de una instancia del sistema, de acuerdo al modelo para él planteado, e igualmente, debe facilitar la mejora del sistema, identificando cuellos de botella y recursos críticos. Para esto, es necesario analizar el sistema bajo diferentes escenarios, con parámetros diferentes cada vez, pero manteniendo la estruc-

tura general del mismo.
Para modelar el desempeño de un sistema software con MARTE, es necesario poner en claro ciertos conceptos que son propios del modelo del dominio, esto es, términos como `BehaviourScenario`, `WorkloadBehaviour`, `Step`, `AnalysisContext`, `Resources`, entre otros.
El `AnalysisContext` combina la representación del sistema de acuerdo a su comportamiento y a la carga de trabajo que tiene que realizar. El contexto de desempeño especifica uno o más `BehaviourScenario` que son utilizados para explorar diferentes situaciones dinámicas que involucran un conjunto específico de recursos. La especificación de un sistema puede contar con varios contextos de desempeño con recursos traslapados, pero un `BehaviourScenario` sólo puede pertenecer a un contexto de desempeño.
Un `BehaviourScenario` describe el comportamiento del sistema en respuesta a un evento disparador. Esta es la unidad básica de comportamiento, y corresponde en UML a un diagrama de comportamiento[7]. Esta compuesto por un conjunto de sub-operaciones denominadas `Step`, las cuales pueden tener predecesor - sucesor. Cada `Step`, para su ejecución, requiere un conjunto de recursos software o hardware. Entre los atributos de `Step` están: la probabilidad de ocurrencia, si es o no síncrono[8], la atomicidad, las repeticiones, la prioridad, y las operaciones externas que afectan el desempeño.
Un contexto de desempeño puede tener cualquier número de `Workload`. Cada `Workload` cuenta con diferentes mecanismos que le permiten realizar las solicitudes al sistema, trabaja con una intensidad de carga diferente y posee sus propios requisitos de QoS. El `Workload` juega el papel de una clase de tráfico que cuenta con mecanismos abiertos o cerrados. El `Workload` abierto es un flujo de eventos que arriban a una tasa determinada de acuerdo a un patrón establecido. Un `Workload` cerrado representa un conjunto fijo de ejecuciones cíclicas activas o potenciales de un `Scenario`.
La Tabla 2.1 muestra la equivalencia existente entre los elementos en la vista del dominio del perfil MARTE y su representación UML.

Vista del dominio	Representación UML
BehaviourScenario	GaScenario
WorkloadBehaviour	GaWorkloadBehaviour
Step	GaStep
AnalysisContext	GaAnalysisContext
Resources	Resource y ResourceInstance
CommunicationChannel	GaCommChannel
CommunicationHost	GaCommHost
CommunicationStep	GaCommStep
ExecutionHost	GaExecHost
ResourcesPlatform	GaResourcesPlatform
WorkloadEvent	GaWorkloadEvent

Tabla 2.1: Equivalencia entre los elementos del perfil MARTE en la vista del dominio y en la representación UML

[7] Diagrama de máquina de estados, diagramas de interacción o diagrama de actividad
[8] Indica si un paso tiene múltiples predecesores pero no múltiples sucesores no probabilisticos

El modelo del dominio para analizar el desempeño de un sistema (véase Figura 2.19), está basado en el modelo de análisis cuantitativo genérico (Generic Quantitative Analysis Modeling – GQAM)[9]. El núcleo del dominio del GQAM, tiene como eje fundamental la descripción de como el sistema, mientras realiza su función, utiliza los recursos.

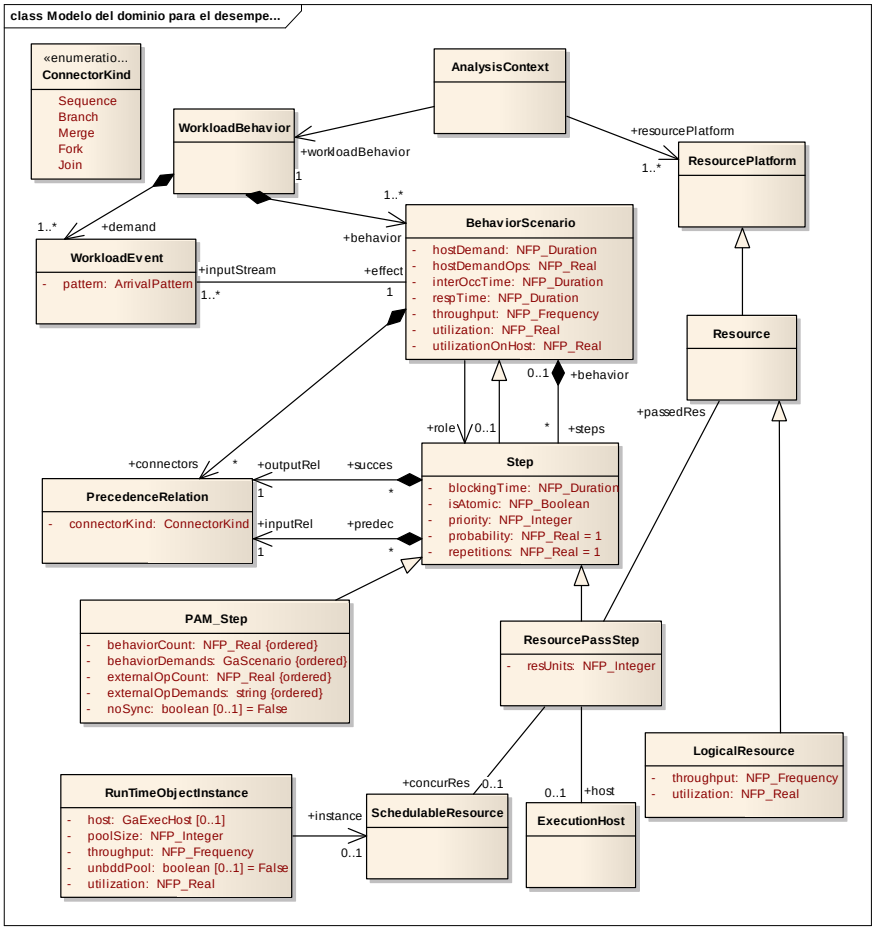

Figura 2.19: Modelo del dominio para el análisis del desempeño (adaptado de [MARTE:2009])

Las técnicas de análisis cuantitativo, determinan propiedades no funcionales (Non-Functional Properties – NFPs), como tiempos de respuesta, plazos no cumplidos, recursos utilizados, tamaño de las colas, entre otras, que asisten a los ingenieros en la toma de decisiones acerca del desempeño del sistema. Aunque el análisis de desempeño se caracteriza por el tiempo que invierte un sistema en desarrollar una tarea y los recursos que utiliza para ello, este trabajo se concentra en los aspectos que tienen que ver exclusivamente con el tiempo.
Un sistema con buen desempeño, debe estar en capacidad de responder, no sólo correctamente a los

[9] Este es un marco de trabajo esencial que define los conceptos básicos de modelado y las propiedades no funcionales de los sistemas de tiempo real embebidos

requisitos que se le hagan, sino también en forma oportuna. Igualmente, debe estar en capacidad de atender requisitos con una frecuencia determinado, bajo condiciones específicas.
Las líneas de tiempo que el sistema debe satisfacer, pueden estar especificadas como plazos duros, suaves o retardos establecidos por una función de costo [Sha:2004]. Los plazos duros, exigen que la respuesta se complete en el tiempo dado. Los plazos suaves, establecen que un porcentaje de las respuestas se completen en el tiempo dado. Los retardos basados en funciones de costo permiten enmarcar el sistema dentro de plazos que, bajo ciertas condiciones, puedan minimizarse. Para este último caso, también son válidas funciones estadísticas que impongan restricciones a los valores promedio de los retrasos, plazos u otras medidas de interés para el desempeño.
Si se quiere medir el desempeño, son muchas las NFPs que se deben incluir (véase la tabla 2.2), sin embargo, el enfoque de este trabajo será sobre las propiedades que tienen que ver con el tiempo y en el marco de los `BehaviorScenario` y los `Step`, esto es: `hostDemand`, `respTime`, `execTime`, `interOccTime`, `throughput`, y `blockingTime`.

NFP	Scenario y Step
repetition:NFP_Real[*]	Número de veces que el paso se repite una vez disparado
probability: NFP_Real[*]	Probabilidad de que el paso sea ejecutado, luego de ejecuta su predecesor
hostDemand: NFP_Duration[*]	La demanda de CPU[10] en unidades de tiempo
hostDemandOps:NFP_Real[*]	La demanda de CPU en unidades de operaciones
priority : NFP_Integer[*]	Para un paso, la prioridad que tiene en el host
respTime: NFP_Duration[*]	Tiempo que tarda la ejecución de un paso o un escenario, desde el evento disparador hasta que termina
execTime : NFP_Duration[*]	respTime menos cualquier retardo debido a la planeación
interOccTime: NFP_Duration[*]	Tiempo que transcurre entre la finalización de un paso o escenario y el comienzo del siguiente
throughput : NFP_Frequency[*]	Frecuencia promedio a la que se inician los paso o escenarios
utilization : NFP_Real[*]	La fracción de tiempo durante la cual el esenario está activo, entre el evento disparador y su finalización
utilizationOnHost: NFP_Real[*]	Fracción de tiempo que el host está ocupado ejecutando el escenario
blockingTime: NFP_Duration[*]	Retardo del paso o escenario

Tabla 2.2: Propiedades útiles para analizar el desempeño (adaptada de [MARTE:2009])

La especificación UML para analizar el desempeño de un sistema software, lo que deja claro con respecto a `BehaviorScenario` y `Step`, es que un conjunto secuencial de `Step` componen un `BehaviorScenario`. Sin embargo, un `Step` puede ser refinado por un `BehaviorScenario`. Más allá de esto, no es posible inferir si un `Step` se corresponde con un componente software, una operación de una clase, con una instrucción de un lenguaje de alto o bajo nivel, o con un ciclo máquina de la CPU. Queda abierto el mapeo de un `Step` a los componentes hardware o software del sistema bajo estudio.

2.6. El lenguaje de especificación de valores

El lenguaje de especificación de valores (VSL – Value Specification Language) es utilizado para especificar valores para las restricciones, las propiedades y los estereotipos relacionados con las NFPs en diagramas UML que utilicen el perfil MARTE. Sin emabrgo, VSL puede ser empleado en cualquier elemento UML en el que se requiera especificar un valor.

2.6.1. Descripción general

La sintaxis abstracta de VSL reutiliza varios conceptos del metamodelo de UML, sin embargo no todo es formalmente importado de allí. La estructura general de paquetes de VSL puede verse en la Figura 2.20, allí se identifican 5 paquetes: `DataTypes`, `LiteralValues`, `Expressions`, `CompositeValues` y `TimeExpressions`. `DataTypes` describe los conceptos que definen la extensión de los tipos de datos de UML utilizados por VSL. `LiteralValues` permite trabajar con valores constantes y tipos primitivos de UML, así como también con reales y fechas. `Expressions` define la estructura de las expresiones, incluyendo las variables y la referencia a valores de un elemento del modelo. `CompositeValues` define 4 tipos de valores compuestos: `interval`, `collection`, `tuple` y `choice`. `TimeExpressions` presenta una sintaxis especializada para expresiones y valores relacionados con el tiempo.

2.6.2. Perfil UML para VSL

No todos los conceptos de VSL establecidos en el modelo del dominio se mapean directamente con estereotipos o valores etiquetados de UML, ya que algunos conceptos se definieron con el fin de ser implementados en un metamodelo separado. Fundamentalmente, los estereotipos de UML que se tomaron con el fin de definir los conceptos de VSL, tienen que ver con la definición de tipos de datos y la declaración de variables (véase Figura 2.21).
`BoundedSubtype` define los subtipos del VSL que se derivan de la metaclase `DataType` de UML. Para estos subtipos debe especificarse el tipo de dato base de UML, el intervalo de posibles valores, y si los límites del intervalo están incluídos dentro del rango de posibles valores del mismo. `IntervalTime` es un tipo de dato compuesto que define un conjunto de valores por medio de dos límites.

```
[100..100E3] # un intervalo numérico simple
[Fmin..Fmax[ # intervalo variable que no incluye a 'Fmax'
```

Nótese que es posible incluir o no lis límites del intervalo.

`CollectionType` describe un conjunto de elementos que son de interés para un contexto en particular. El tipo de los elementos de `CollectionType` no tiene restricción y debe ser declarado por medio de `CollectionAttribute`. Esto permite que un `CollectionType` incluya otros

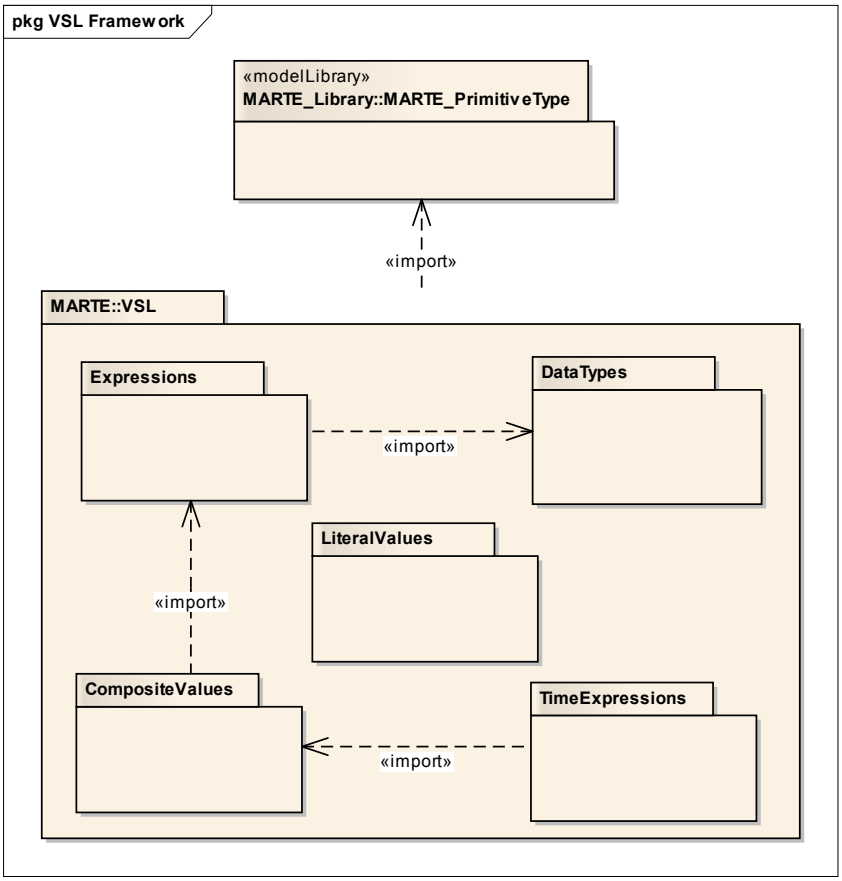

Figura 2.20: Estructura del marco de trabajo de VSL [MARTE:2009]

`CollectionType`, permitiéndose anidaciones de profundidad arbitraria en las colecciones. Por ejemplo:

```
{10,25,33,109,200} # es una colección de números
{'procesador', 'memoria', 'BIOS'} # es una colección de strings
{{0,20,198},{1,2}} # colección de colecciones
```

`TupleType` permite combinar varios tipos de datos en un sólo tipo compuesto, algo parecido a lo que es una estructura en un lenguaje como C. Las tuplas se identifican por sus tributos: un nombre y un tipo. Una tupla puede contener otras tuplas o incluso colecciones. Por ejemplo:

```
{Freq=1.2E3,unit=KHz} # tupla con nombre para los valores
{103, 95, 99} # tupla sin nombre para los valores
```

`ChoiceType` combina diferentes tipos en un sólo tipo de datos, algo parecido a una unión en un lenguaje como C. Los atributos de `ChoiceType` permiten identificar el tipo, tamaño, unicidad y orden de las diferentes alternativas del tipo de dato. Igualmente se puede definir el dato seleccionado por defecto.

```
periodic(period=10, jitter=0.1) # e l ChoiceType se denomina
                                  'periodic', cuyo tipo de
                                  dato es una tupla, con los
                                  valores 'period' y 'jitter'
```

`Var` permite crear variables con nombre, tipo de dato y naturaleza (entrada, salida, entrada/salida).

```
out$var1 # declaración de una variable de salida
var1 # llamado a la variable
```

`ExpressionContext` es un contenedor en el cual se definen las variables. De esta forma los conflictos entre variables globales pueden resolverse utilizando una expresión de llamada de variables en forma completa, por ejemplo:

```
ModulacionAnaloga::FM::Beta
```

En este caso `ModulacionAnaloga` es el contexto, `FM` es un subcontexto, y `Beta` es la variable. `ExpressionContext` sólo tiene un atributo, de tipo string, el cual identifica el nombre del contexto.

2.7. El lenguaje de restricción de objetos

OCL es un lenguaje formal que permite escribir restricciones sobre modelos UML, haciendo a estos más precisos y menos ambiguos, de tal forma que se logre una mayor refinación del modelo. Por ser una adición estándar a UML, está completamente integrado con este y permite consultar los valores de los elementos de un modelo y establecer restricciones sobre ellos. Sin embargo, con OCL no es posible cambiar el valor de un elemento del modelo, ya que es enteramente un lenguaje de consulta, como SQL [OCL:2010]. Las expresiones OCL se pueden asociar a un modelo UML como notas en el diagrama correspondiente, o en un archivo adjunto.
OCL es un lenguaje de especificación puro, cuyas expresiones no tienen efectos colaterales, esto es, la evaluación de una expresión OCL simplemente retorna un valor, y no cambia nada dentro del modelo.
OCL no es un lenguaje de programación, ya que con él no es posible escribir lógica de programas o control de flujo. Sólo se pueden activar operaciones de consulta y no invocar procesos. Ya que inicialmente OCL fue concebido como un lenguaje de modelado, sus expresiones, por definición, no son directamente ejecutables.
Cada expresión en OCL tiene un tipo, por lo tanto sus expresiones deben seguir las reglas de conformación de tipos del lenguaje. Por ejemplo, no se puede comparar un entero con un string. Cada clasificador definido dentro del modelo UML representa un tipo OCL. Adicional a esto, existe un conjunto de tipos predefinidos en la especificación del lenguaje.
Ya que OCL es un lenguaje de especificación, las implementaciones están fuera del alcance del lenguaje, y por lo tanto no pueden ser expresadas con este. La evaluación de una expresión OCL es instantánea, esto quiere decir, que el estado de un objeto en un modelo no puede cambiar durante la

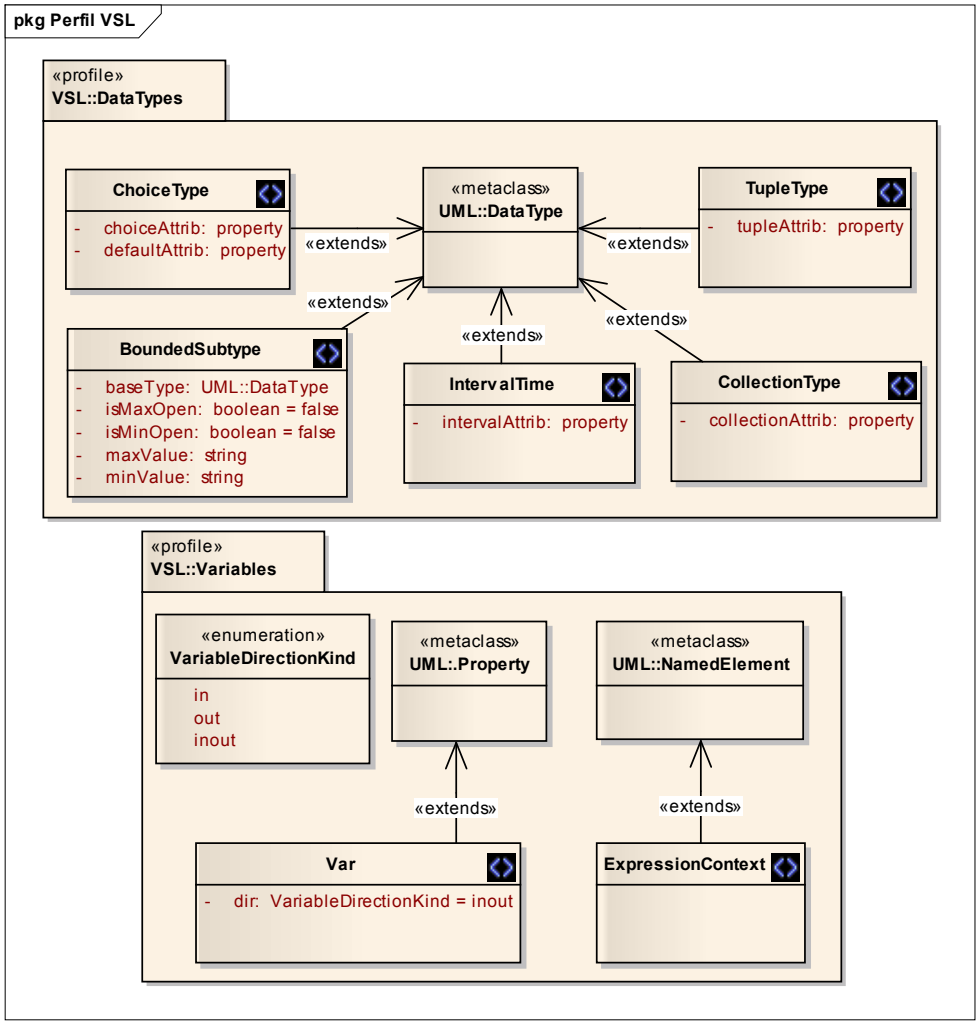

Figura 2.21: Extensión UML para definición de tipos de datos y variables [MARTE:2009]

evaluación de la expresión.
Todo lo anterior permite pensar en el uso de OCL como un lenguaje para:

- Consultas
- Especificar invariantes sobre clases y tipos en el modelo de clases
- Especificar tipos invariantes para estereotipos
- Describir pre y post-condiciones sobre operaciones y métodos
- Especificar objetivos para los mensajes y las acciones
- Especificar restricciones sobre operaciones
- Especificar reglas de derivación para los atributos o expresiones sobre modelos UML

Capítulo 3
Antecedentes

Resumen Este capítulo presenta un caso de estudio de un sistema embebido de tiempo real – RT/E, el cual consiste del sistema de vigilancia de un campamento militar por medio de una red inalámbrica de área personal – WPAN. Este sistema, se especifica utilizando diversos enfoques encontrados en la literatura, entre ellos están: el marco de trabajo basado en requisitos no funcionales – NFRs, y una variante que incluye el lenguaje de modelado unificado – UML; la especificación actual de UML; una solución basada en el lenguaje de restricción de objetos – OCL; UML con el soporte del perfil para modelado y análisis de sistemas embebidos de tiempo real – MARTE; soluciones que mapean NFRs a UML; el uso de ontologias para la representación de NFRs; la combinación de aspectos con NFRs; y la representación con el lenguaje Markup extendido – XML. A cada solución presentada se le destacan sus bondades y falencias. Finalmente se realiza un análisis de los problemas encontrados con los NFRs: problemas de definición, problemas de clasificación y problemas de representación.

Los sistemas software, están caracterizados por la funcionalidad requerida y los requisitos globales que se les impone. Estos últimos, juegan un rol crítico durante el desarrollo e implementación del sistema, y si no son tenidos en cuenta, la experiencia dicta que son de los errores más costosos y difíciles de corregir una vez el sistema ha sido implementado.
Sorpresivamente, estos requisitos globales, denominados requisitos no funcionales – NFRs, han recibido poca atención en la literatura y debido a esto, los ingenieros de software no le han dado la importancia que si han recibido otros requisitos que verdaderamente son menos críticos. La RE ha tratado, generalmente, a los NFRs con declaraciones informales, debido en parte a que pueden ser contradictorios, difíciles de satisfacer y bastante más difíciles de validar.
Como se trató en la sección 2.2.1, existe un consenso bastante amplio acerca de lo que son los requisitos funcionales, conservando dos tendencias: la primera es haciendo énfasis sobre la función [Xu:2005] [Chung:2000] [IEEE:1990] [Robertson:1999] [Sommerville:2007], la segunda hace énfasis sobre el comportamiento [Anton:1997] [Davis:1993]. Existe sólo una dificultad entre las diferentes definiciones, y son los requisitos de tiempo. Algunos podrian considerarlos elementos claves del comportamiento, otros los consideran requisitos de desempeño, los cuales hacen parte de los NFRs.
Los NFRs no cuentan con el consenso con el que cuentan los RFs, por lo tanto existen múltiples

maneras de definirlos, clasificarlos y representarlos.
Para evaluar las formas de trabajo que actualmente se tienen con NFRs, partiremos del sistema de seguridad de un campamento militar como caso de estudio. Las fases previas de educción de requistos se trabajará de acuerdo a UN-METODO [Arango:2006].

3.1. Caso de estudio: sistema de vigilancia para un campamento militar

Consideremos un sistema de seguridad para un campamento militar. Este sistema de seguridad tiene como núcleo principal una red inalámbrica de sensores, en adelante denominada red inalámbrica de área personal (Wireless Personal Area Network – WPAN). Los nodos de la WPAN son de dos clases, coordinadores y dispositivos finales, cada WPAN puede sólo tener un coordinador, pero múltiples dispositivos finales. El coordinador se encarga de iniciar la WPAN, asignar las direcciones a cada dispositivo final, servir como puente de comunicación entre nodos finales si fuera necesario, aceptar, rechazar o cancelar la vinculación de un dispositivo final a la WPAN por él administrada, entre otras funciones. Los dispositivos finales tienen asociados detectores de movimiento o sensores de calor, los cuales permiten determinar la presencia de intrusos. Esta información debe ser enviada al coordinador con el fin de que éste la centralice y tome la decisión de activar o no la alarma general de seguridad del campamento, la cual está asociada a otro dispositivo final.
El sistema de seguridad es un sistema de tiempo real, ya que tiene varias exigencias que se enuncian a continuación:

- Se considera que un dispositivo final está activo cuando cualquiera de los sensores a él adjunto se activa
- El coordinador consultará cíclicamente a los dispositivos finales acerca de su estado. El tiempo que transcurre entre una consulta y otra no debe ser mayor a $1s$
- Los dispositivos finales consultarán sus sensores sólo a petición del Coordinador
- Luego de detectada la presencia de intrusos por parte del dispositivo final, el sistema debe garantizar que la alarma se active en menos de $500ms$

La topología de la WPAN puede verse en la Figura 3.1. Aquí se distribuye la red entre un nodo coordinador, y cuatro dispositivos finales; tres utilizados para sensar y un cuarto para activar la alarma. El sistema sólo posee detectores de movimiento y sensores térmicos, ambos infrarrojos. Un dispositivo final cuenta con un sensor y un detector, mientras que los otros sólo cuentan o con un sensor o con un detector.

3.2. Diferentes soluciones al sistema de vigilancia para un campamento militar

Existen múltiples propuestas para trabajar con NFRs, algunas se enfocan en la etapa de educción de requisitos, definiendo ontologias y modelos del dominio adecuados a este tipo de restricciones; otras metodologías se centran en definiciones de arquitecturas que soporten de forma adecuada las características transversales de los NFRs, en algunos casos la filosofía de diseño es objetual en

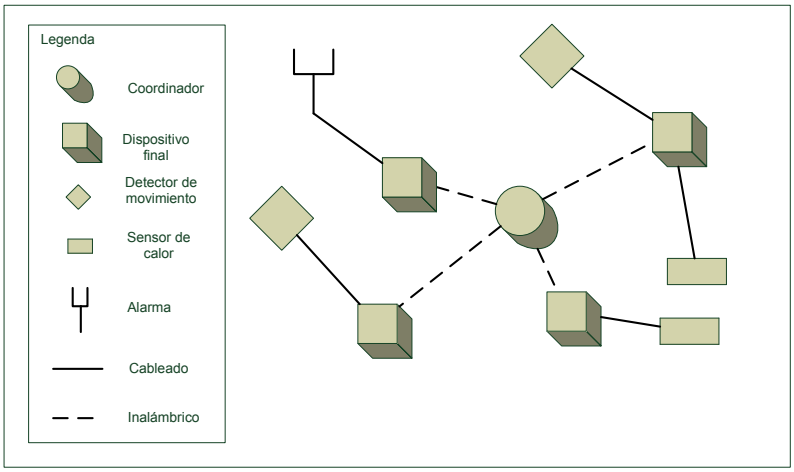

Figura 3.1: Topología de la WPAN del sistema de vigilancia del campamento militar

otros es dirigida por aspectos; los lenguajes formales, como OCL, son una alternativa adicional para especificar esta clase de requisitos. Este capítulo, presenta diferentes metodologías de trabajo para especificar y representar NFRs, basado en el caso de estudio presentado en la sección 3.1. Inicialmente se presenta una educción de requisitos, que en forma general, puede servir como punto de partida para las diferentes metodologías de solución presentadas en esta tesis.

3.2.1. Educción de requisitos del sistema de vigilancia para un campamento militar

El diagrama de procesos (véase Figura 3.2) identifica tres roles, donde el rol coordinador es el encargado de realizar la mayoría de los procesos, los otros dos roles son de dispositivos finales, donde tres dispositivos finales se encargan de monitorear las áreas en las que está dividido el campamento, y el cuarto se encarga de activar la alarma general.

La tabla explicativo de los procesos (véase Tabla 3.1) permite indicar, para cada proceso, el objetivo y la duración y frecuencia con que se realiza.

3.2.2. Solución: marco de trabajo basado en NFRs

Esta propuesta de solución se apoya en [Mylopoulos:1992], [Chung:1995], [Chung:2000] y [Supakkul:2005].

A diferencia de los enfoques de diseño dirigidos por la funcionalidad, es posible utilizar un marco de trabajo basado en NFRs con el fin de dirigir el proceso de diseño completo de un sistema. Esto obliga al desarrollador a trabajar con los NFRs en mente todo el tiempo. Los principales pasos de este proceso de desarrollo pueden verse en la Figura 3.3. Estos pasos no son necesariamente secuenciales y se puede iterar sobre ellos.

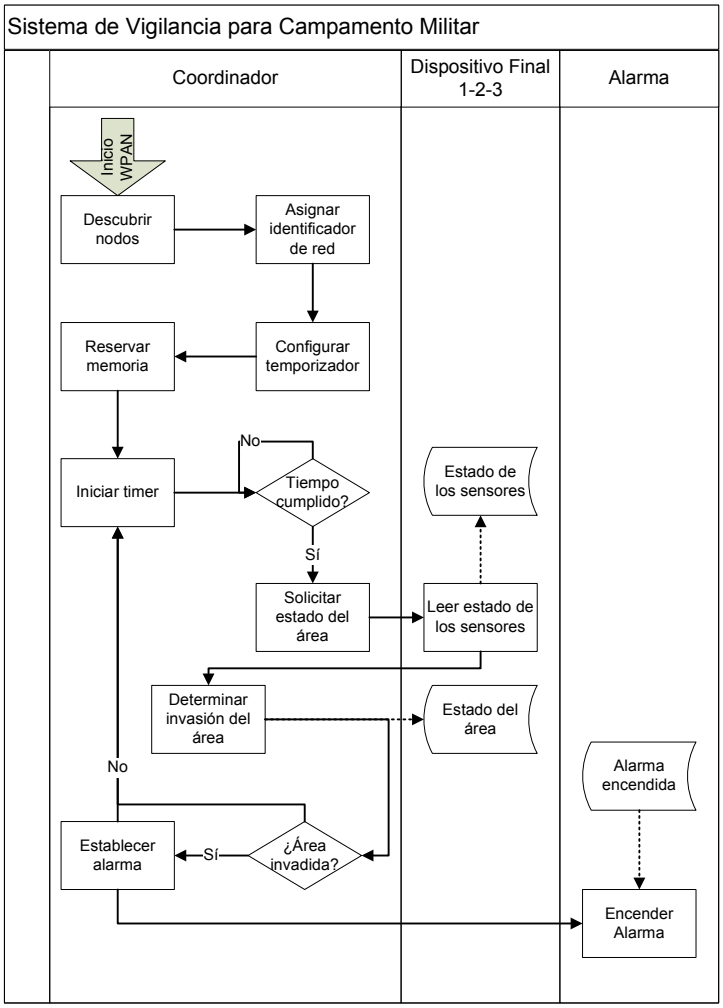

Figura 3.2: Diagrama de procesos del sistema de vigilancia del campamento militar

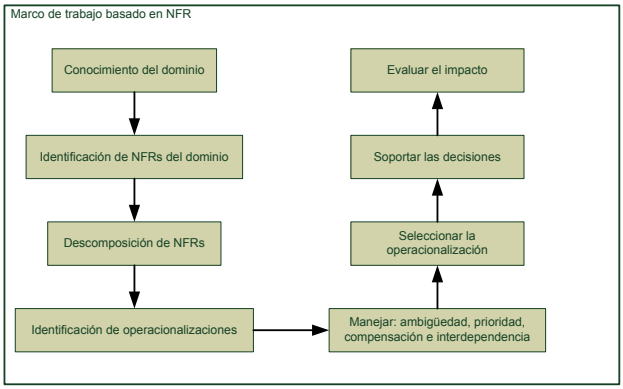

Figura 3.3: Actividades realizadas en el marco de trabajo basado en requisitos no funcionales

Nombre	Objetivo	Duración/Frecuencia
Descubrir nodos	Permitir que nuevos dispositivos finales sean detectados por el coordinador	Se realiza una vez, cada que se inicia el sistema
Asignar identificador de red	Asignar a cada nuevo dispositivo final detectado un identificador de red único	Se realiza una vez, cada que se inicia el sistema
Configurar temporizador	Establecer el tiempo que servirá de base para escanear toda la WPAN	Se realiza una vez, cada que se inicia el sistema
Reservar memoria	Reservar espacio de memoria suficiente para almacenar la información proveniente de todos los nodos finales y poder detectar intrusos	Se realiza una vez, cada que se inicia el sistema
Iniciar timer	Comenzar el proceso de escaneo de la WPAN	Se realiza cada segundo
Solicitar estado del área	Permite consultar a cada dispositivo final, responsable de un área, el estado de sus sensores	Se realiza tres veces por segundo
Leer estado de los sensores	Leer el estado de los sensores y detectores asociados a cada dispositivo final	Se realiza una vez por segundo, para todos los sensores asociados al dispositivo final
Desterminar invasión del área	De acuerdo a la información recibida por el coordinador de parte de los dispositivos finales, este proceso permite determinar si en el área se encuentran intrusos	Se realiza tres veces por segundo
Establecer alarma	Determinar el estado en el que debe estar la alarma para el siguiente escaneo de la WPAN	Se realiza cada segundo
Encender alarma	La alarma la enciende el dispositivo final que la tiene asociada	Se realiza cada que se detecta intrusos en un área

Tabla 3.1: Tabla explicativa de los procesos de la WPAN

Para elicitar correctamente los NFRs es necesario que el desarrollador adquiera un conocimiento profundo del dominio de la aplicación, esto incluye ítemes como RFs, flujo de trabajo principal de la organización y prioridades de la organización.
Inicialmente, es necesario identificar los NFRs principales que el sistema bajo desarrollo debe cumplir. Estos NFRs son tratados como softgoals por alcanzar. Un softgoal, no necesariamente tiene un criterio absolutamente claro de satisfacibilidad. Aunque algunos pueden ser medidos y cuantificados, un enfoque cualitativo se utiliza cuando se está explorando posibles alternativas de desarrollo.
El desarrollador debe identificar técnicas específicas, que permitan elegir soluciones que cumplan los requisitos previamente definidos. Por lo tanto, será necesario descomponer de manera sistemática los softgoals iniciales en unos más específicos subsoftgoals.
Los NFRs tienen un tipo que indica en particular de que NFR estamos hablando, por ejemplo, seguridad y desempeño son dos tipos de softgoal. Se hace necesario dividir los softgoals en otros más simples, donde se trate la ambigüedad, y se vuelva importante la información del dominio y la priorización. A medida que avanza este análisis, se vuelve clave tener en cuenta la interdependencia entre softgoals, lo que permitirá sintetizar soluciones con calidad para el sistema bajo consideración. Son varias las alternativas que se tendrán, de las cuales una debe elegirse pero tomando decisiones racionales. Finalmente, se hace necesario evaluar que también los requisitos principales se han cumplido.
El marco de trabajo basado en NFRs, ofrece una estructura para representar y registrar el proceso

de diseño y razonamiento en grafos, llamado grafos de interdependencia softgoal (SIG – Softgoal Interdependency Graphs). Los SIG, permiten visualizar la elaboración, el análisis y la revisión del sistema de una forma incremental e interactiva, ya que registra las consideraciones del desarrollador y muestra la interdependencia entre softgoals.

La mayoría de los conceptos del framework aparecen en forma gráfica en un SIG. Los softgoal, los cuales son "soft" por naturaleza, se muestran como nubes etiquetadas con un tipo y un tópico. Los requisitos principales se muestran como softgoals en la parte superior del gráfico. Los softgoals se conectan por medio de enlaces de interdependencia, los cuales se muestran como líneas. Los softgoals tienen etiquetas asociadas[1], las cuales sirven para soportar los procesos de razonamiento durante el diseño. Las interdependencias muestran el refinamiento de los softgoal padres en softgoals más específicos, los hijos. También muestran la forma como los softgoals hijos contribuyen a que el softgoal padre se cumpla.

Con el fin de utilizar esta metodología, el diagrama de objetivos de la WPAN será reemplazado por un SIG. Partiendo del softgoal seguridad en el campamento, `Seguridad[campamento]`[2], se van indicando los demás softgoal que los subrogan (véase Figura 3.4). Para este caso se tienen seguridad en cada sector, `Seguridad[sector]` softgoal muy crítico, confiabilidad del Coordinador, `Confiabilidad[coordinador]` softgoal muy crítico, confiabilidad de la alarma, `Confiabilidad[alarm]` softgoal crítico y desempeño de la WPAN, `Desempeño[wpan]` softgoal crítico. De acuerdo a los demás softgoals presentes, se puede determinar el grado de satisfacibilidad de cada softgoal, lo cual se representa con un símbolo dentro de la nube. Los softgoal se puede satisfacer completa o parcialmente, igualmente se puede negar completa o parcialmente, puede entrar en conflicto con otro u otros softgoal, no haberse aun decidido su grado de satisfacibilidad o desconocerse.

El grado de satisfacibilidad que tiene un softgoal, está relacionado con la contribución que los softgoal que lo subrogan le aportan. Un signo $+$ o $++$ en el enlace indica que el softgoal fuente aporta positivamente a la realización del softgoal destino, mientras que un signo $-$ o $--$ es todo lo contrario.

Analicemos el softgoal crítico desempeño de la WPAN, `Desempeño[wpan]`. Este softgoal tiene contribuciones de otros tres softgoal: seguridad de la información, `Seguridad[info]`, desempeño del coordinador, `Desempeño[coordinador]` y desempeño del dispositivo final, `Desempeño[dispFinal]`. Satisfacer los softgoal de desempeño para el Coordinador y el Dispositivo Final, `Desempeño[coordinador]` y `Desempeño[dispFinal]`, contribuye a satisfacer el softgoal de desempeño de la WPAN, `Desempeño[wpan]`, mientras que satisfacer el softgoal de seguridad de la información, `Seguridad[info]`, perjudica el desempeño de la WPAN.

La seguridad de la información se considera un softgoal operacionalizable[3] y se descompone en dos softgoal operacionalizables, la integridad y la confidencialidad de la información. La integridad se satisface por la tecnología utilizada para la red inalámbrica, la confidencialidad se satisface encriptando la información, actividad que disminuye el desempeño debido a la utilización de tiempo y recursos. Por lo tanto, el hecho de no satisfacer la confidencialidad genera que el softgoal `Seguridad[info]` sea denegado, y como este contribuye nega-

[1] valores que representan el grado en el cual un softgoal se alcanza

[2] Tipo: Seguridad, tópico: campamento, softgoal muy crítico

[3] El símbolo es una nube dibujada con una línea más oscura

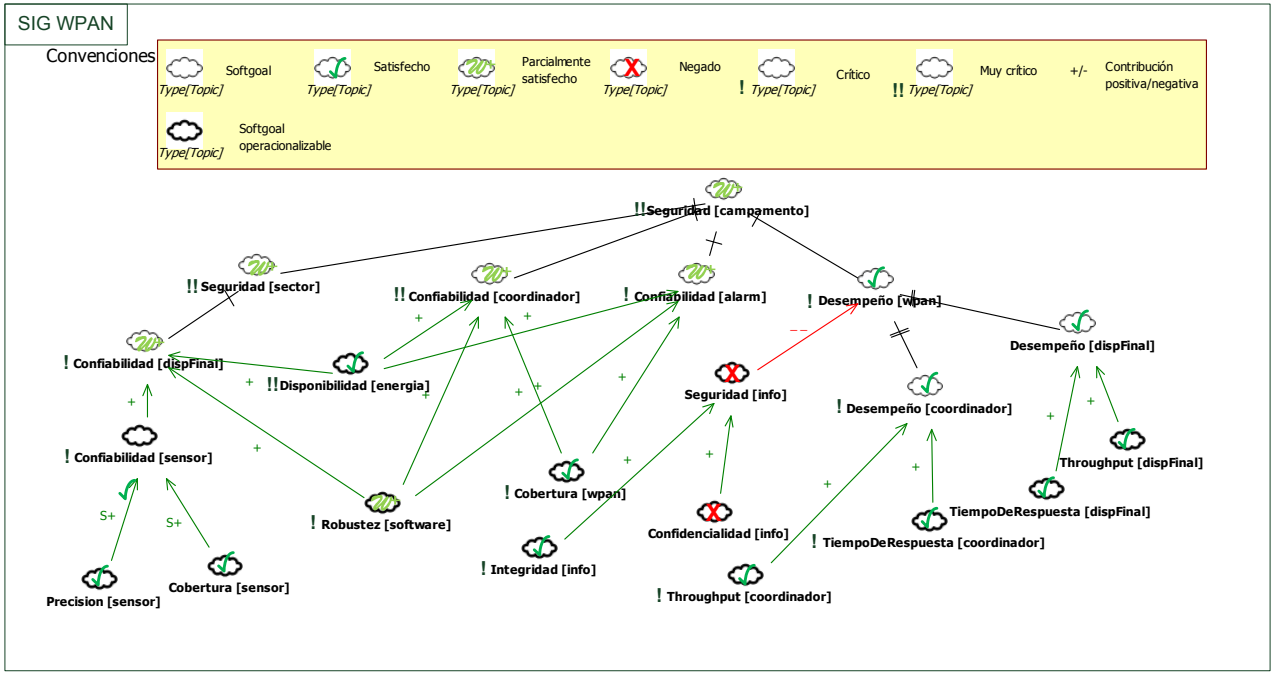

Figura 3.4: SIG del sistema de vigilancia del campamento militar

tivamente a Desempeño[wpan], entonces en general el desempeño se verá beneficiado. El desempeño del coordinador y el dispositivo final se subrogan a los softgoal Throughput y TiempoDeRespuesta, los cuales son softgoal operacionalizables.
Análisis similar se puede realizar con los demás softgoal, los cuales se llevan hasta el punto de lograr softgoal operacionalizables. Obsérvese que el softgoal Disponibilidad[energia], que está catalogado como muy crítico, contribuye positivamente a alcanzar los softgoal con tipo Confiabilidad y tópicos alarm, dispFinal y coordinador. Robustez[software] contribuye positivamente a la confiabilidad del dispositivo final y el coordinador, sin embargo, el grado de satisfacibilidad es parcial. Cobertura y Precision del sensor son softgoal que contribuyen a la confiabilidad del sensor, el cual a su vez contribuye a la confiabilidad del dispositivos final.
Para continuar con el modelado se hace necesario establecer una relación entre el marco de trabajo basado en NFRs y un lenguaje de modelado, como UML. En [Supakkul:2005] se crea un perfil UML, denominado perfil UML Softgoal, que consiste de dos perfiles: SIG y Procedure.
El perfil SIG representa los conceptos del marco de trabajo basado en NFRs que se utilizan en los SIG (véase Figura 3.5), ya que extiende los artefactos UML con el fin de representar un modelo basado en Softgoal o Contribution. Estos dos últimos conceptos son una especialización completa y disyunta de Proposition, esto es, no hay otra especialización posible para Proposition.

El metamodelo de UML no define una metaclase para la asociación entre actor y caso de uso, por lo tanto un actor no puede asociarse a ningún elementos del modelo, incluídos los NFRSoftgoal. Para esto, los autores de esta metodología proponen adicionar una nueva metaclase al metamodelo de UML, llamada Actor-UsecaseCommunication, la cual hereda de la metaclase AssociationClass, y se relaciona con las metaclases Actor y Usecase.
El perfil Procedure (véase Figura 3.6) trata con los conceptos que tienen que ver con métodos, reglas de correlación, y procedimientos de evaluación. Se introduce en este perfil los conceptos de ProcedureRepository y Procedure como extensiones a las operaciones y las clases, lo que permite utilizar el lenguaje de restricción de objetos (Object Constraint Language – OCL), para especificar restricciones sobre los diferentes estereotipos definidos en el perfil SIG.

De acuerdo a los perfiles propuestos el SIG representado en la Figura 3.4, correspondiente al sistema de seguridad del campamento militar, puede representarse según la porción del diagrama de clases de la Figura 3.7.
El acople entre clases y estereotipos permite representar los conceptos de los perfiles antes planteados. NFR1 es el nombre de la clase raiz del sistema con el estereotipo NFRSoftgoal, representa a Seguridad[campamento], el cual está compuesto por cuatro softgoal más a través de una relación AND, denominada Decomposition1 con estereotipo DecompositionCT. Los cuatro softgoal descendientes son NFR2, NFR3, NFR4 y NFR5. El tipo Seguridad es compartido por los softgoal NFR1 y NFR2, mientras que el tipo Desempeño por NFR3 y NFR5.

Debido a que el diagrama de clase se crece bastante con esta representación, otro fragmento, fruto de los perfiles arriba definidos, puede verse en la Figura 3.8. En este diagrama de clases se puede observar la representación de los softgoal operacionalizables, y la contribución de estos a la satisfacción de los softgoal padres. Por ejemplo, Seguridad[info] es un softgoal operacionalizable, identificado por el estereotipo OperationalizingSG, descompuesto en los softgoal operacio-

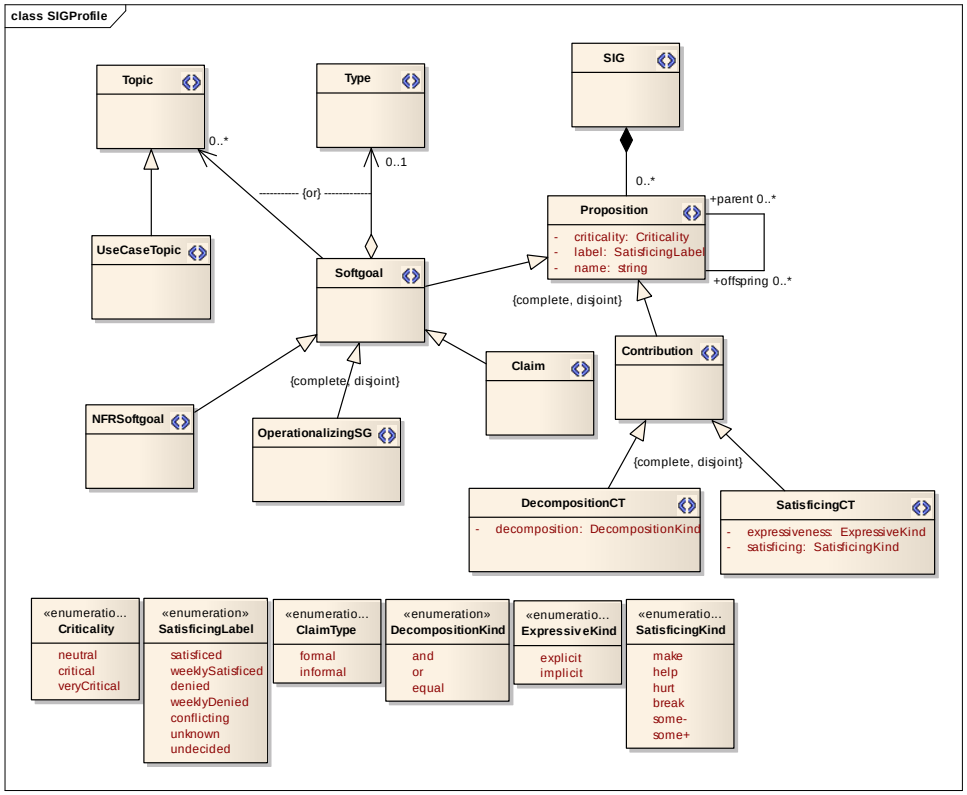

Figura 3.5: Perfil SIG [Supakkul:2005]

nalizables `Confidencialidad[info]` e `Integridad[info]`, los cuales tienen contribuciones de tipo `help` a `Seguridad[info]`.
Estos perfiles permiten definir de forma precisa la sintaxis visual y la semántica de los conceptos modelados, las relaciones entre tales conceptos, y las restricciones sobre los conceptos individuales y sus relaciones.

En el marco de trabajo basado en NFRs, los softgoals representan los NFRs. Refinándolos gradualmente, es posible representar adecuadamente el sistema, pasando de softgoals de muy alto nivel a softgoals operacionalizables. En el SIG obtenido, si bien es posible representar, de forma cuantitativa, algunos requisitos de tiempo con softgoal de afirmación, no es posible cuantificar la satisfacibilidad del NFR desempeño, ya que las contribuciones de cada NFR al desempeño del sistema están definidas en forma cualitativa.
El SIG permite manifestar de manera explícita los conflictos y la sinergia entre los diferentes NFRs, además facilita la toma de algunas decisiones de diseño con el fin de lograr la satisfacibilidad de los objetivos o la compensación en algunos debido al sacrificio en otros. Todo esto se apoya en la priorización que se le da a cada softgoal.
El SIG resulta ser la materia prima para comenzar a especificar el sistema en un lenguaje de

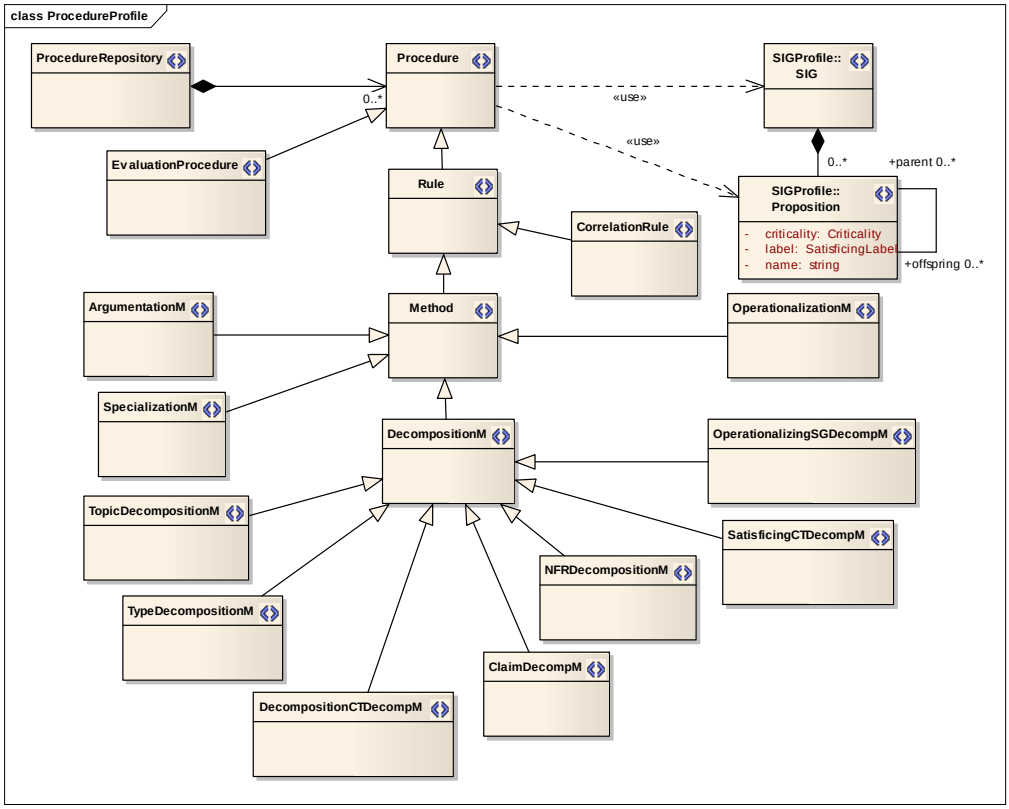

Figura 3.6: Perfil Procedure [Supakkul:2005]

modelado que lleve la especificación hasta el plano de la implementación. Si se utiliza UML con la extensión propuesta en [Supakkul:2005], se tienen dificultades con el enfoque cualitativo del SIG, con la diversidad de NFRs que no alcanzan a cubrirse con los métodos de descomposición, y con el tamaño del diagrama de clases obtenido, que para sistemas de mediana complejidad es inmenso.
Sin embargo, a pesar de todo esto el marco de trabajo basado en NFRs es una herramienta que puede ser muy útil como complemento a UML a la hora de especificar NFRs temporales.

3.2.3. Soluciones basadas en UML

Aunque UML está más asociado con modelar sistemas de software orientados a objetos, tiene una aplicación mucho más amplia debido a sus mecanismos de extensibilidad: restricciones, estereotipos y valores etiquetados.
Para manejar las restricciones se definió OCL [OCL:2010], mientras que las propiedades no funcionales se pueden representar con el perfil MARTE para UML.

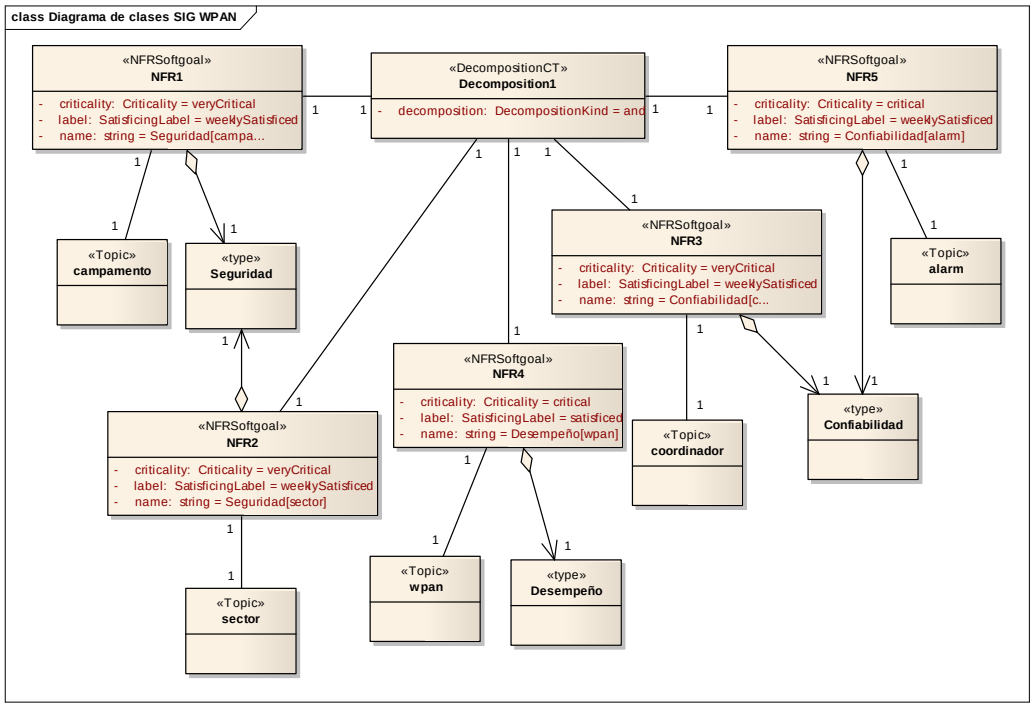

Figura 3.7: Porción del diagrama de clases para el SIG de la WPAN

3.2.3.1. Solución: UML 2.2

Esta solución se basa en [UMLSuper:2009]. Como se vió en la sección 2.4.2, son múltiples los diagramas UML que pueden utilizarse para el modelado de un sistema, sin embargo, generalmente no todos entregan información útil que amerite incluirlos dentro del modelo que se está desarrollando. Debido a la clase de sistema que nos ocupa, lo especificaremos con 6 diagramas UML: casos de uso, actividades, clases, secuencias, tiempo y despliegue.

Un elemento clave en la educción de requisitos son los casos de uso, ya que se consideran un excelente punto de partida a la hora de desarrollar, diseñar, probar y documentar un sistema software orientado a objetos. El diagrama de casos de uso se considera que debe ser uno de los primeros planos del sistema que se deben tener, ya que es imprescindible saber lo que el sistema debe hacer antes de comenzar su diseño.

Para la WPAN construiremos tres diagramas de casos de uso, uno para el coordinador, otro que aplica para cada uno de los tres dispositivos finales que incluyen sensores y detectores de presencia, y el último para el dispositivo final que controla la alarma.

El coordinador cuenta con un conjunto de casos de uso que le permiten configurar la WPAN (véase Figura 3.9), entre ellos están el descubrimiento de nodos y la asignación de ID a cada dispositivo final que se una a la WPAN. La planificación del escaneo de la WPAN y el manejo de los recursos hardware, memoria y timer, son otros elementos necesarios para la correcta administración de la WPAN.

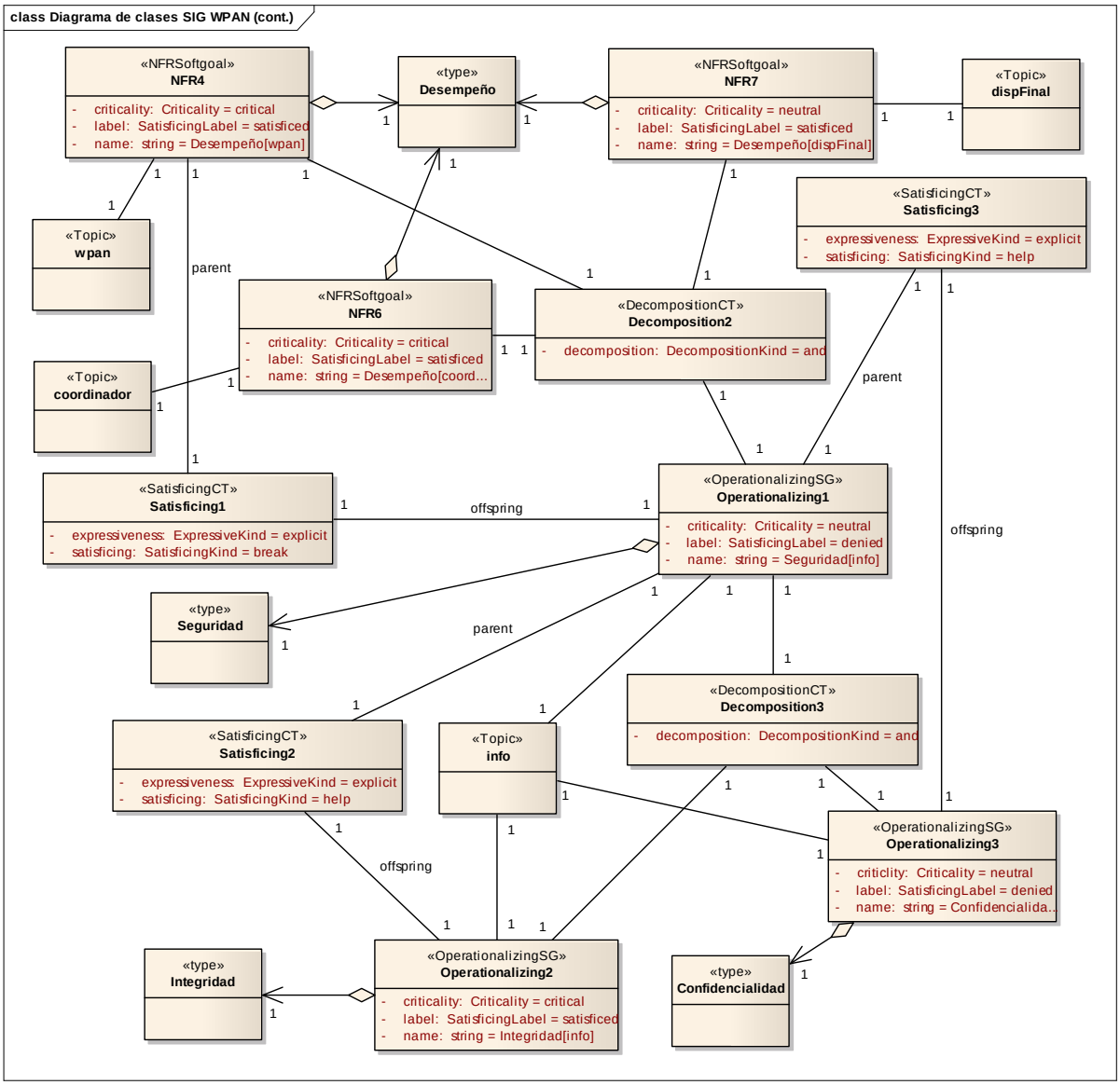

Figura 3.8: Continuación del diagrama de clases para el SIG de la WPAN

El aseguramiento del área, requiere de la colaboración de varios casos de uso, como son, el planificador de escaneo de la red, la solicitud a los dispositivos finales para que realicen medidas de presencia de intrusos en los sectores bajo su control, la determinación de invasión o no del área y el proceso de encendido de la alarma luego de detectar intrusos en el campamento.
Por último, existen un par de casos de uso transversales a toda la WPAN, y son aquellos que soportan la transmisión y recepción de mensajes, y el encapsulamiento de la información en la trama definida de acuerdo al perfil aplicado.

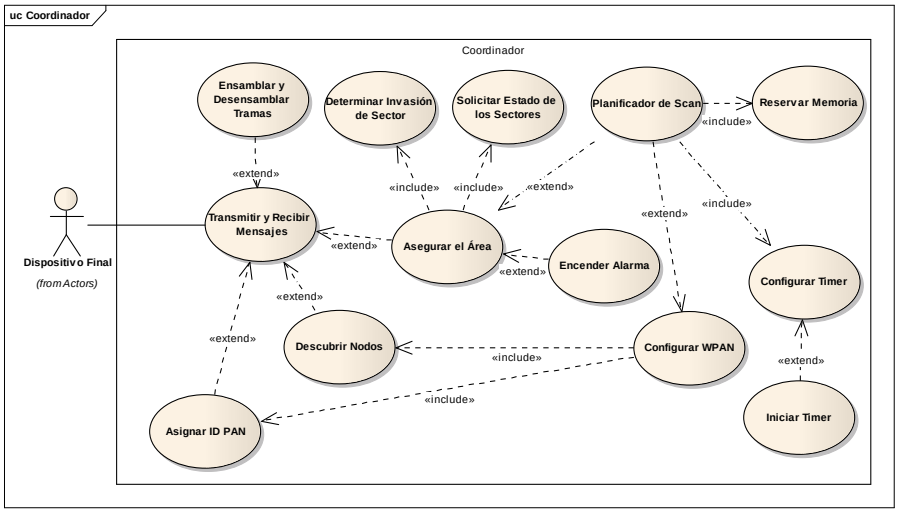

Figura 3.9: Diagrama de casos de uso para el coordinador de la WPAN

El único actor involucrado en este diagrama, es el dispositivo final con el cual se debe comunicar el coordinador.

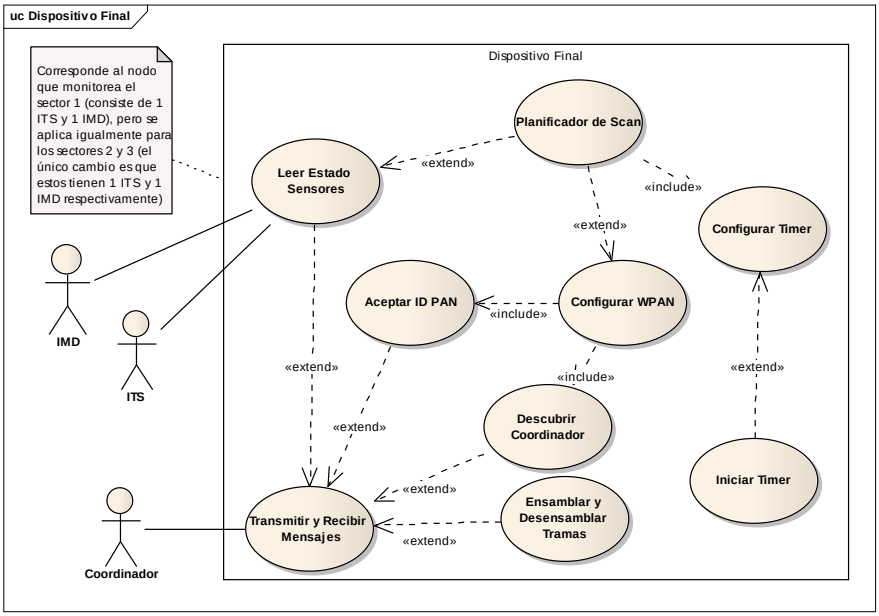

Figura 3.10: Diagrama de casos de uso para los dispositivos finales que incluyen sensores y detectores de presencia de la WPAN

Los dispositivos finales cuentan, igualmente, con casos de uso que les permiten administrar su ingreso a una WPAN, esto es, actividades de búsqueda de coordinador y aceptación de ID asignado por este último (véase Figura 3.10). La lectura de los sensores adjuntos al dispositivo final, es coor-

dinada por un planificador de escaneo, el cual se apoya en recursos hardware como el timer.
En este diagrama de casos de uso, es posible identificar los casos de usos para transmisión y recepción de información, y el encapsulamiento de la misma.
Los actores en este diagrama son el coordinador, que es con el único que se comunica el dispositivo final, y el sensor y detector de intrusos, si los tuviera, que son utilizados para propósitos de medición en el sector que le corresponde vigilar al nodo.

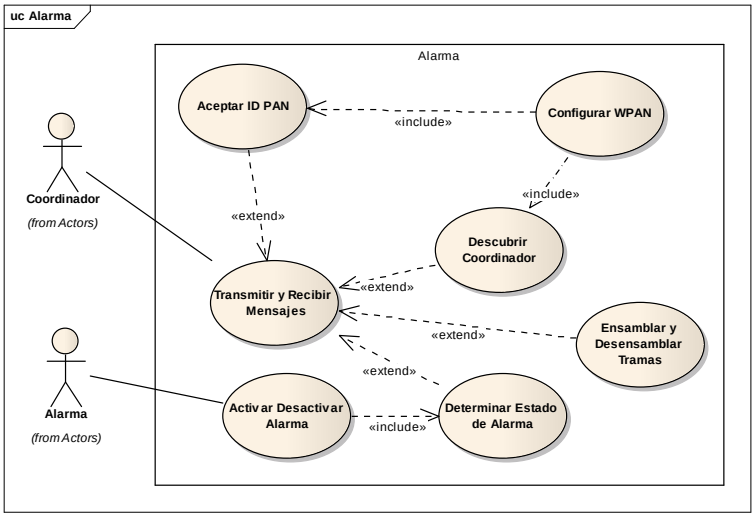

Figura 3.11: Diagrama de casos de uso para el dispositivo final que controla la alarma de la WPAN

El diagrama de casos de uso para la alarma (véase Figura 3.11), es básicamente el mismo que tienen los dispositivos finales, ya que la alarma, en términos de este tipo de WPAN, es también un dispoitivo final. La diferencia de este diagrama de casos de uso, es que existe un actor más, la alarma, y se tienen los casos de uso para actuar sobre ella.
El siguiente diagrama con el que trabajaremos es el de actividades, este diagrama reemplaza, en varios aspectos, al diagrama de procesos (véase Figura 3.2). El diagrama de actividades permite visualizar acciones de alto nivel, las cuales se encadenan con el fin de representar los procesos que ocurren en el sistema (véase sección 2.4.2.2). El diagrama de actividades se presenta en la Figura 3.12, allí se pueden identificar tres roles: Coordinador, DispFinal 1-2-3 y Alarma.

El Coordinador tiene tres actividades, una de configuración de la WPAN, `WPANSetup`, otra de planificación de la WPAN, `Scheadulability` y la última para activar la alarma, `setAlarm`. Igualmente tiene una actividad estructurada repetitiva, `loop`, donde el cuerpo consiste en la consulta a cada dispositivo final. Nótese que la actividad `getValueArea` debe esperar a que la actividad `getValueSensor`, ejecutada por los dispositivos finales, le retorne el control para poder finalizar. Luego de conocer el estado de cada sector, la actividad estereotipada como `test` es la que decide si la consulta a los dispositivos finales se termina o no. Si se finaliza el lazo es para activar la alarma, si no, se continua con la consulta a los dispositivos finales. Los roles `DispFinal1-2-3` y `Alarma` sólo se modelaron con una actividad, para leer los sensores o para activar la alarma.
No existe facilidad para expresar duración de una actividad, de tal forma que los NFRs tempo-

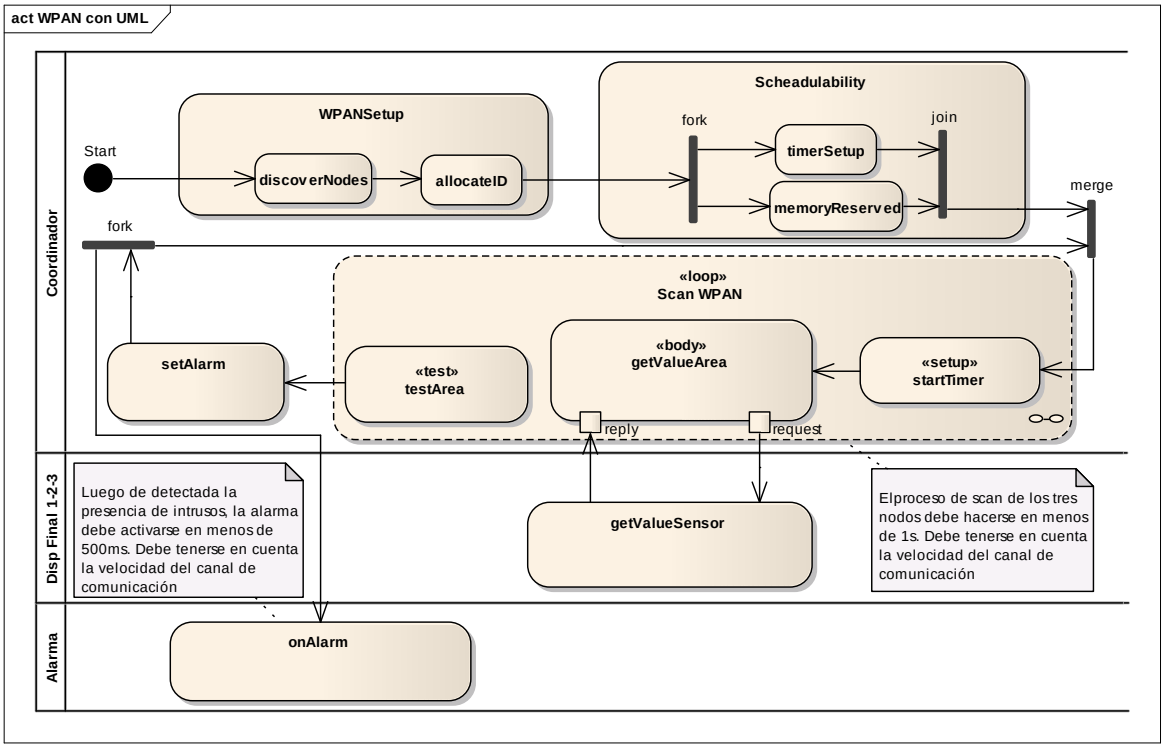

Figura 3.12: Diagrama de actividades UML de la WPAN

rales identificados en el ejemplo no pueden fácilmente expresarse en esta clase de diagrama. Por ejemplo, el NFR temporal que exige que cada segundo se consulten todos los sectores para determinar si existen intrusos, tiene que ver con las actividades `getValueArea`, `testArea` y `getValueSensor`, ya que la duración de estas tres actividades es la que determina si los tres sectores pueden ser consultados en menos de 1*s*. El NFR que exige la activación de la alarma en menos de 500*ms*, tiene que ver con las actividades anteriores más `setAlarm` y `onAlarm`. Debe tenerse en cuenta que el canal de comunicaciones también juega un papel central para que estos NFRs puedan satisfacerse, ya que velocidades de transmisión muy bajas harán imposible alcanzar el rendimiento esperado.

La única forma de expresar los NFRs temporales de los que habla el ejemplo, es por medio de notas, como la que está enlazada a la actividad estructurada estereotipada como `loop`, que indica, en lengujaje natural, la restricción temporal de las actividades y la advertencia acerca del canal de comunicaciones. Igualmente se tiene una nota enlazada con la actividad `onAlarm`.

El diagrama de clases para el Coordinador se muestra en la Figura 3.13. Se pueden identificar dos clases centrales: `Planificador` y `BuilderFrame`. La primera de las clases se encarga de manejar los tiempos, con la clase `Timer`, configurar la WPAN, con la clase `Wpan` y detectar intrusos en el campamento militar, con la clase `IntrusionDetection`.

El proceso de transimisión tiene como eje central a la clase `BuilderFrame`, la cual se encarga de ensamblar las tramas antes del proceso de transmisión y desensamblar las tramas luego del pro-

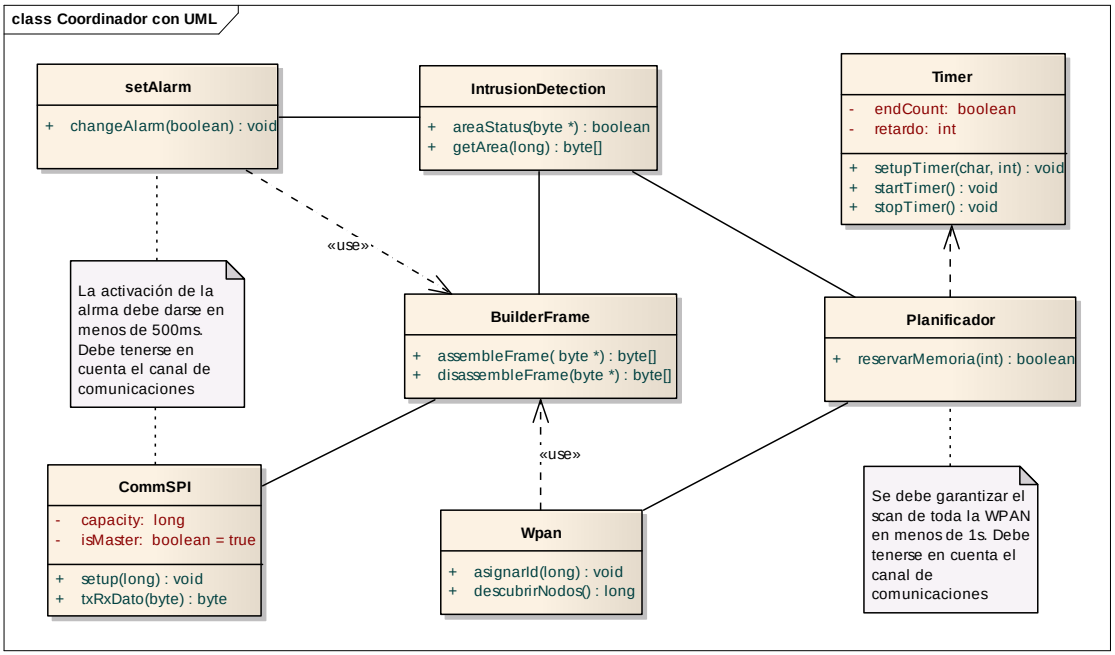

Figura 3.13: Diagrama de clases UML de la WPAN

ceso de recepción. Esta clase se vale del módulo SPI (Serial Peripheral Interface), representado por la clase `CommSPI`, la cual abstrae el módem inalámbrico que tiene cada nodo de la WPAN.
Al igual que en el diagrama de actividades, en el diagrama de clases las restricciones se ponen como notas en lenguaje natural o estructurado, como OCL, pero de manera formal no es posible representar NFRs temporales.
Las interacciones en un sistema se pueden representar en diagramas de secuencia. Como se vio en la sección 2.4.2.2 los diagramas de secuencia capturan el orden de las interacciones entre las partes del sistema.
En el diagrama de secuencia del Coordinador (véase Figura 3.14) identificamos siete participantes, los cuales corresponden a instancias de las clases definidas en el diagrama de la Figura 3.13. Este diagrama tiene varias fases: la primera es la fase de configuración de la WPAN, la cual se da sólo una vez y antes del ciclo infinito; la segunda es el ciclo infinito, en el cual se quedá el sistema con el fin de garantizar la seguridad de la WPAN; la tecera es el ciclo denominado `scan`, el cual garantiza que los tres nodos serán consultados, condición escrita de la misma forma que se escriben condiciones en los ciclos for en C; la última fase es un fragmento opcional, que sólo se ejecuta si se activa un sensor o detector en el sistema.

Debido a la interacción de los nodos de la WPAN, el diagrama de secuencia completo es muy grande, por lo tanto se utilizan referencias a los diagramas de secuencia de los dispositivos finales y de la alarma. Por asuntos de espacio y porque su aporte no es muy significativo, no se muestra el diagrama de secuencia de la alarma, pero si se muestra el diagrama de secuencia típico para los dispositivos finales (véase Figura 3.15).
En el diagrama de secuencia para los dispositivos finales se cuenta con siete participantes, seis de

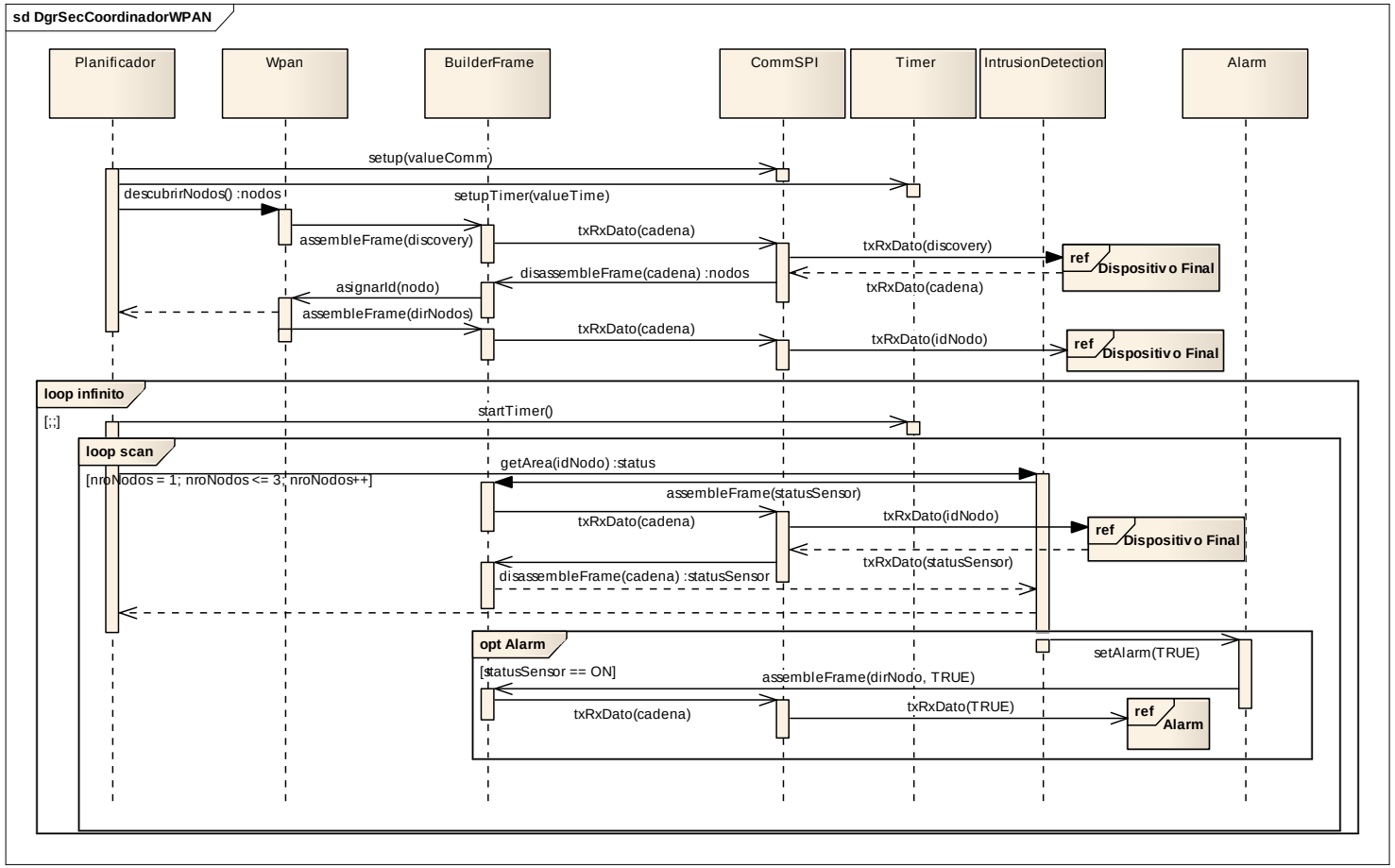

Figura 3.14: Diagrama de secuencia del Coordinador de la WPAN

ellos son instancias de clases del sistema, y el participante restante representa al sensor o detectar conectado al dispositivo final. En este diagrama se pueden identificar tres fases: una de configuración, durante la cual el dispositivo final se tiene que comunicar con el Coordinador; un ciclo infinito, en el cual se espera la solicitud de lectura de los sensores o detectores por parte del Coordinador; y un ciclo opcional, el cual se ejecuta sólo si la solicitud del Coordinador es de chequeo de las zonas.

De una manera muy informal, en este caso por medio de una nota, se indica la restricción temporal que garantizará el escaneo de la WPAN en menos de $1s$.

Las restricciones temporales son adecuadamente presentadas en el diagrama de tiempo de UML. El caso de estudio descrito en la sección 3.1 tiene un requisito temporal, que reproducimos aquí: ´´El coordinador consultará cíclicamente a los dispositivos finales acerca de su estado. El tiempo que transcurre entre una consulta y otra no debe ser mayor a $1s$". El diagrama de secuencia representa fielmente la tarea cíclica exigida por el requisito, pero enfocándose en el orden secuencial de los mensajes. Mientras tanto, el diagrama de tiempo permite expresar la restricción cuantitativa de que la consulta a los tres nodos no debe tardar más de $1s$.

En la Figura 3.16 se muestran seis participantes en el diagrama de tiempo para el Coordinador de la WPAN. El planificador, por ejemplo, cuenta con cinco estados, uno de espera, necesario en casi cualquier participante, dos de configuración, no cíclicos, uno de consulta a los dispositivos finales `checkSectores` y otro para activar la alarma, `setAlarm`, en el hipotético caso que se detecten intrusos.

Cada que se está consultando un dispositivo final, el planificador pasa al estado `checkSectores`, mientras que los demás participantes comienzan a pasar, cada uno, a los estados necesarios que contribuyen a la actividad propuesta. A cada estado se le puede asignar una duración, como en el ejemplo que se indica $220ms$ para `checkSectores` y $80ms$ para `setAlarm`. Con esto se garantiza que el escaneo de toda la red, y la posible activación de la alarma producida por alguno de los dispositivos finales, tendrá que realizarse en $300ms$, lo que a su vez permite establecer el throughput de la WPAN a por lo menos $3Hz$.

Se muestran dos mensajes cuyo origen es el participante `Planificador`, los cuales sirven para iniciar la configuración del módulo SPI, mensaje `SetupSPI`, y para que el timer inicie la medición del tiempo del escaneo de los tres nodos de la WPAN, mensaje `StartTimer`.

Después del estado `setupWPAN`, el participante `Planificador` pasa al estado de espera `idle`, durante el cual los participantes `BuildeFrame` y `Modem` se encargan de transmitir la dirección asignada a cada nodo. Al momento de pasar al estado `idle`, el `Planificador` envía el mensaje de inicio del timer, `StartTimer`, el cual activará la medición del tiempo para garantizar el escaneo de la WPAN en menos de $1s$.

El participante `BuilderFrame` sólo tiene dos estados, uno para ensamblar o desensamblar tramas, y el otro de espera. Dado que se utilizará en la implementación una tecnología estándar, el proceso de manipulación de las tramas siempres se considera igual, ya sea para ensamblar o desensamblar. La mayoría del tiempo este participante se encuentra en estado de espera.

`CommSPI` va de la mano de `BuilderFrame`, ya que este último utiliza a `CommSPI`, en unos casos, para transmitir las tramas ensambladas, mientras que en otros casos recibe de este las tramas que el Coordinador requiere desensamblar.

El `Modem` representa los tiempos durante los cuales se está transmitiendo la información o uno de los nodos finales está procesando información para enviarla al Coordinador. Estos tiempos se refle-

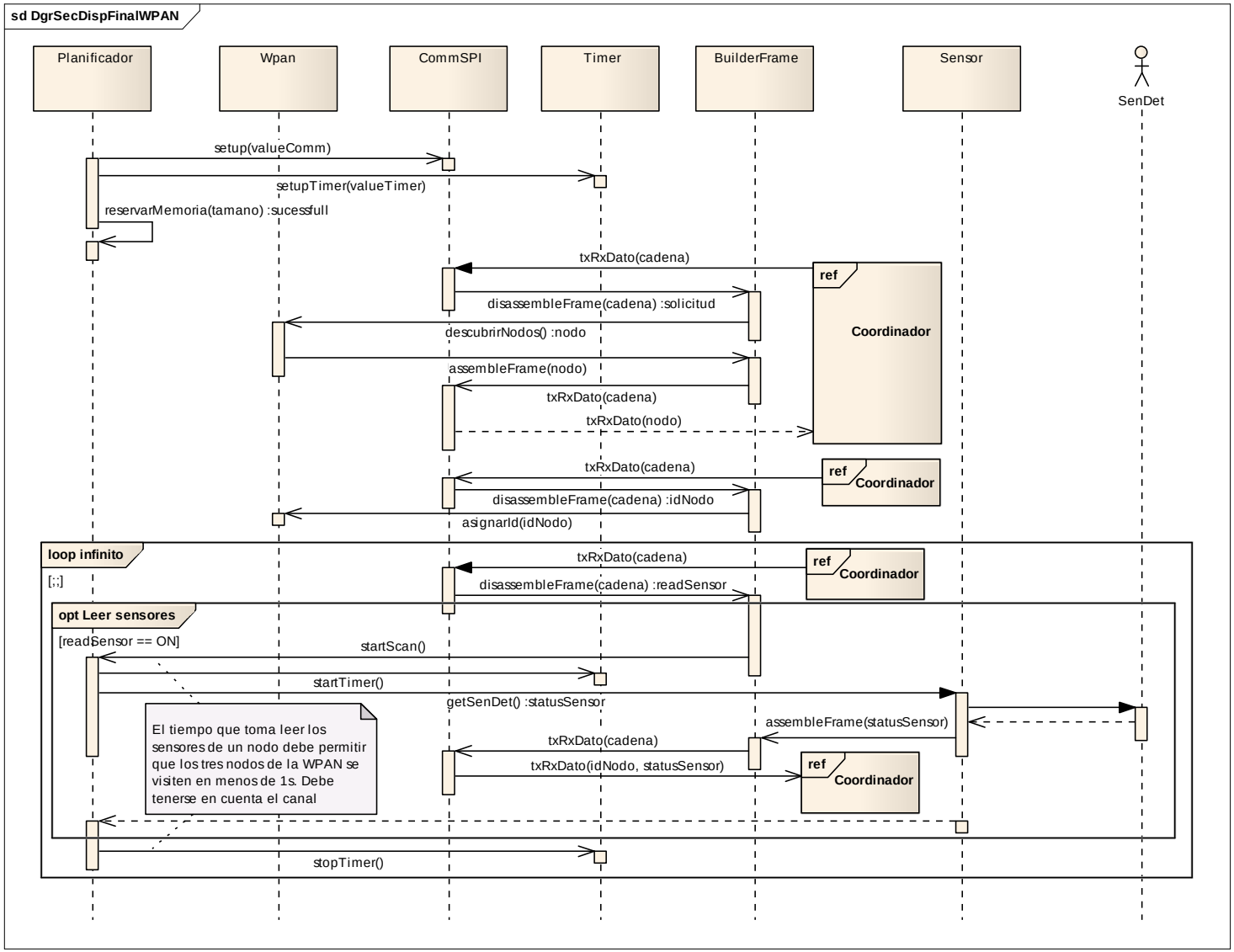

Figura 3.15: Diagrama de secuencia del dispositivo final de la WPAN

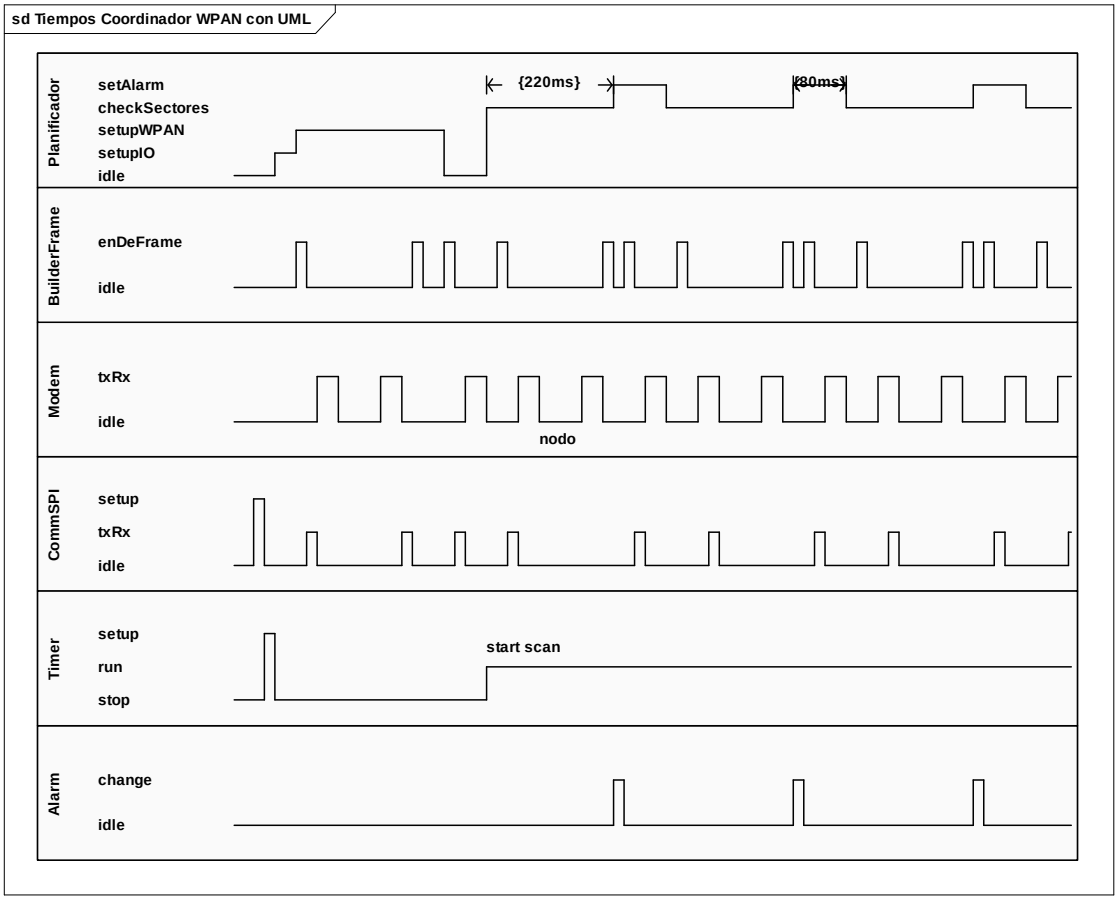

Figura 3.16: Diagrama de tiempo del Coordinador de la WPAN

jarían en los diagramas de tiempo de los nodos finales, los cuales no se muestran en este informe por cuestiones de espacio.

El `Timer` recibe un mensaje del `Planificador`, `start timer`, con el cual inicia el escaneo de la WPAN. Cada cierto tiempo, menor a $1s$, el `timer` vuelve y comienza la cuenta, estado `reset`. De esta forma, se garantiza un throughput de por lo menos $3Hz$ en el escaneo de la WPAN.

El participante `Alarm`, se mantiene la mayor parte del tiempo en estado de espera, `idle`, y sólo pasa a estado de conmutación de alarma, `change`, cada que un sensor se encuentre activo en alguna de las tres áreas en las cuales se dividió el campamento.

Para visualizar como el software que se está ejecutando se despliega sobre el hardware físico y como se conecta ese hardware, utilizamos el diagrama de despliegue (véase Figura 3.17).

El `Coordinador` y el `DispFinal` son nodos en los cuales se ejecuta el software del sistema. En ambos nodos se incluyen básicamente los mismos artefactos: uno para planificar las actividades, otro de transmisión y recepción de mensajes, un tercero de temporización y el cuarto es la pila ZigBee que contiene todo el protocolo de comunicación.

El canal de comunicaciones está modelado como otro nodo, el cual se estereotipa como IEEE

802.15.4, que es el estándar que especifica las características eléctricas, mecánicas, de direccionamiento físico y acceso al medio necesarias para la comunicación entre los nodos. Los dispositivos finales pueden tener conectados los sensores o la alarma, ambos a través del bus de datos, denominado aquí como `Backend`.

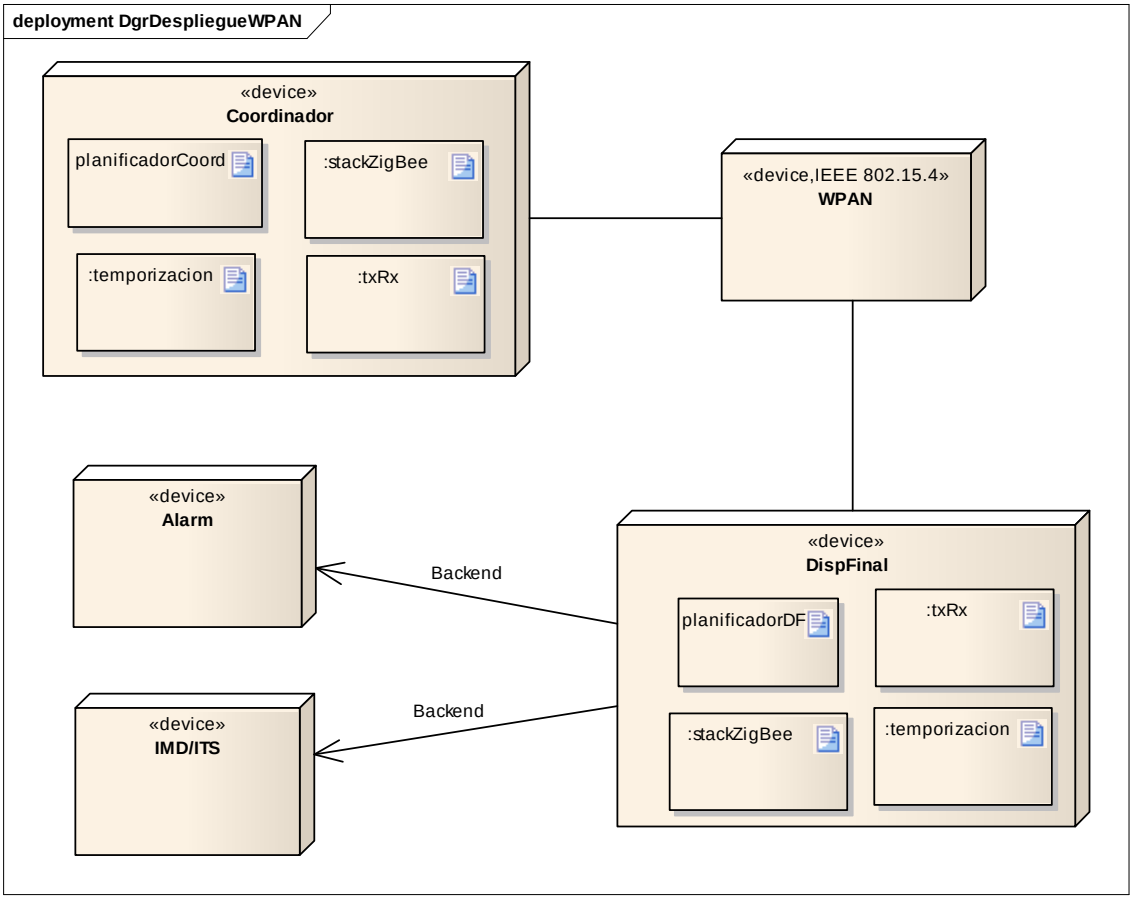

Figura 3.17: Diagrama de despliegue de la WPAN

UML como tal es un lenguaje para modelar sistemas, lo que lo convierte en practicamente un lenguaje de propósito general. Por la diversidad de NFRs que existen, y los diferentes tipos de sistemas sobre los que se requiere especificarlos, es obligatorio especializar el lenguaje para satisfacer todos los matices de los sistemas involucrados. En el caso de NFRs temporales sobre sistemas RT/E, se requieren especializaciones del tiempo y de los recursos, que no se encuentran en la especificación de UML. Existen algunos diagramas, como el diagrama de tiempo y el de despliegue, que permite representar algunos componentes requeridos por los sistemas RT/E, pero nunca con el nivel de detalle que exige una especificación para esta clase

de sistemas. Por ejemplo, tamaño de memoria, hilos, ciclos de CPU, tiempo de respuesta, rendimiento, son algunas características que no se pueden especificar con UML clásico.

3.2.3.2. Solución: OCL 2.2

Como se vio en la sección 2.7, OCL fue introducido para soportar la especificación de restricciones sobre diagramas UML y principalmente utilizado para formular invariantes y pre y postcondiciones sobre las operaciones. Aunque OCL también se aplica en diagramas de comportamiento, por ejemplo como condiciones de guarda en diagramas de secuencia o estado, no es posible, sin embargo, especificar restricciones concernientes a comportamientos dinámicos y propiedades temporales en tales diagramas. A pesar de ello, existen varios trabajos que acercan OCL a la especificación de propiedades no funcionales temporales.
En [Flake:2002a] se define un perfil UML a partir de una extensión temporal a la versión 2.0 de OCL [Flake:2002]. Esto permite volver a OCL aplicable para especificaciones de tiempo real, ya que se introducen operaciones temporales que permiten a los modeladores especificar comportamientos orientados a estados. Esta extensión razona acerca de posibles estados futuros de los objetos, donde la definición de la semántica se basa en una variante de CTL limitada en el tiempo.
Se realiza un fuerte trabajo de profundización en el concepto de estado, configuración de estado y secuencia en la configuración de estado. Proponen la inclusión de nuevas metaclases al metamodelo de tipos de OCL, `ConfigurationType` y `PathType`. La primera metaclase permite establecer la configuración válida de un estado, esto es, conocer el conjunto de estados simples que componen un estado activo. `PathType` permite modelar las trayectorias de ejecución de un diagrama de estados, esto es, las secuencias de configuraciones válidas por las que pasa el diagrama.
La metaclase `TemporalExp` se incluye en el metamodelo de expresiones de OCL, y permite representar expresiones temporales relacionadas con una trayectoria de ejecución. Esta es la superclase abstracta de `PastTemporalExp` y `FutureTemporalExp`, las cuales son expresiones temporales orientadas al pasado y al futuro respectivamente.
Para la sintáxis se introduce una nueva expresión temporal, `post(a,b)`, y un operador, @, que es utilizado sólo para dichas operaciones temporales. `@post(a,b)` retorna un posible conjunto de trayectorias de ejecución futuras en el intervalo `[a,b]`, incluyendo la configuración en los puntos `a` y `b`. La semántica para esta operación puede representarse así:

```
obj@post(a:integer, b:OclAny) :
           Set(Sequence(Set(OclState)))
pre: a >= 0 and ((b.oclIsTypeOf(Integer)
and b >= a) or (b.oclIsTypeOf(String)
and b = ´inf’))
```

Esta definición indica que la operación devuelve un conjunto de estados por los cuales pasa el objeto entre un intervalo de tiempo dado por `[a, b]`, donde `b >= a`, `a` es un entero y `b` puede ser un entero o infinito.
De acuerdo al diagrama de estados de la WPAN (véase Figura 3.18), el NFR descrito en la sección

3.1, donde se exige que todos los nodos de la red (3 en total) sean consultados en menos de 1*s*, puede representarse utilizando la siguiente expresión:

```
context Coordinador
inv: Planificador@post(1, 333) --> forAll
     (p:OclPath | p->includes(checkSectores))
```

Esta expresión retorna el conjunto de estados por los cuales el sistema debe pasar en menos de 333*ms* y que incluyen al estado `checkSectores`. De esta forma, se obtiene la ruta que sigue el sistema cuando se envia un mensaje de consulta del estado de un sector y se recibe la correspondiente respuesta. Para nuestro caso la consulta retornará los estados `checkSectores`, `enDeFrame`, `txRx` de `CommSPI` y `txRx` de `Modem`, y todos los estados del dispositivo final involucrados en la respuesta. Se logra de esta forma, limitar la duración en tiempo del conjunto de estados necesarios para ejecutar una acción o proceso dentro del sistema.

En [Lavazza:2005] se propone OTL, el cual es una extensión de lógica temporal a OCL. OTL proporciona los operadores básicos de la lógica temporal, esto es, `Always`, `Sometimes`, `Until`, etc. Igualmente, OTL permite razonar acerca del tiempo en forma cuantitativa. OTL extiende la librería estándar de OCL [OCL:2010] adicionando tres nuevas clases: `Time`, `Duration` e `Interval`. La clase `Time` modela los instántes de tiempo, la clase `Duration` modela la duración de un intervalo de tiempo, por ejemplo la distancia entre dos instántes de tiempo. De esta forma, una instancia de la clase `Interval` puede ser definida por su instante de tiempo inicial y su duración. Estas clases heredan directamente de la clase `OclAny`, la cual es la clase raiz de la jerarquía de tipos de OCL.
Las clases `Time` y `Duration` tienen definidas operaciones que permiten el manejo adecuado del tiempo. Por ejemplo, la clase `Time` tiene asociada una operación que permite determinar si un instante de tiempo antecede o precede a otro; otra operación para hallar la distancia entre dos instantes de tiempo, retornando un objeto del tipo `Duration`; y otra que toma un parámetro *d*, de tipo `Duration`, y retorna un objeto de tipo `Time` que se encuentra a una distancia *d* en el tiempo. Por otro lado, la clase `Duration` tiene operaciones de suma y resta entre sus objetos, mientras que la clase `Interval` tiene operaciones para determinar si dos intervalos se traslapan, si un intervalo contiene a otro, y para unir dos intervalos traslapados. De esta manera, es posible que los modeladores usen tiempo cuantitativo. Debe tenerse en cuenta que no existen restricciones acerca del tipo de tiempo a utilizar, esto es, `Time` y `Duration` pueden ser discretos o densos.
De acuerdo al diagrama de estados de la Figura 3.18, la sentecia OTL para el NFR que exige que todos los nodos de la red sean consultados en menos de 1*s* será:

```
context Coordinador
let X: Duration = [0, 333] in
     X --> exists(p:OclPath | p
     -> includes(checkSectores))
```

Esta expresión indica que el conjunto de estados por los cuales el sistema debe pasar en 333*ms*, debe incluir al estado `checkSectores`. De esta forma, se garantiza que el sistema envió un mensaje de consulta del estado de un sector y recibió la correspondiente respuesta. Se logra de esta forma, que el sistema pase por el estado `checkSectores`, el cual indica que se realizó la consulta a uno de los sectores.

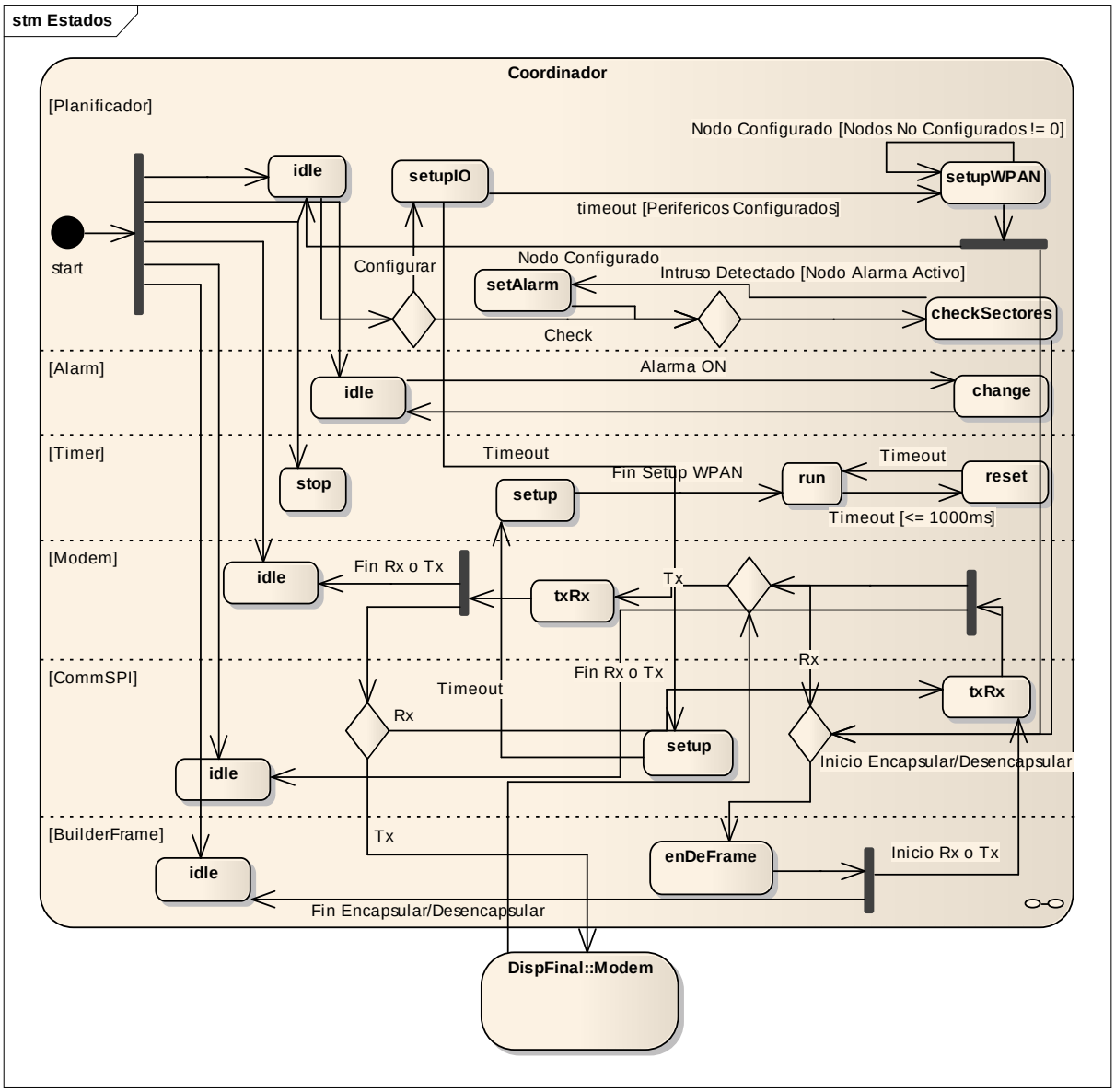

Figura 3.18: Diagrama de estados del Coordinador de la WPAN

En los diagramas UML se pueden representar restricciones en cualquier lugar donde sea válido el uso del término `expression`, pero esta restricción puede o no ser estructurada. La manera más fácil de hacerlo es por medio de lenguaje natural, pero la impresición e informalidad de este tipo de expresiones las hace difíciles de satisfacer.
OCL no cuenta con una librería que permita representar expresiones temporales, las cuales pudieran ser adicionadas a un diagrama UML como restricciones en un lenguaje estructura-

do y formal. A pesar de que OTL no exige cambios al metamodelo de OCL, ya que `Time`, `Duration` e `Interval` son sólo tres nuevos tipos que se adicionan a la librería estándar de OCL como especializaciones de `OclAny`, esta representación de NFRs es específica para diagramas de estado, que no siempre son útiles y necesarios al momento de modelar algunos sistemas. Otro inconveniente que presenta esta forma de representar NFRs es que OCL tiene una notación difícil de seguir, dificultando la labor del ingenierp de software.

3.2.3.3. Solución: UML 2.2 con MARTE 1.0

Esta solución se basa en [MARTE:2009], el cual especifica un perfil UML para sistemas embebidos de tiempo. El diagrama de despliegue del sistema puede verse en la Figura 3.19 (no se especifican los artefactos en cada nodo debido a que ya se presentaron en la figura 3.17). Aquí se observa que el contexto de desempeño del sistema hace énfasis, utilizando VSL [MARTE:2009], en la cantidad de coordinadores, `in$NCoord`, dispositivos finales `in$NDispFinal`, detectores de movimiento, `in$NIMD`, y sensores de calor, `in$NITS`; así mismo, indica el tamaño máximo de la trama de comunicación entre el coordinador y los dispositivos finales, `in$maxSizeMsg`. El estereotipo que reune la especificación del contexto de desempeño, `<<GaAnalysisContext>>`, tiene la siguiente forma:

```
<<GaAnalysisContext >>
{contextParams={in$NCoord, in$NDispFinal,
in$NITS, in$NIMD, in$maxSizeMsg},
paramValues={1,4,2,2,(128,Bytes)}}
```

En esta especificación se indica que el contexto de despliegue del sistema tendrá 1 coordinador, 4 dispositivos finales, 2 sensores de calor, 2 detectores de movimiento y la trama de comunicación entre el coordinador y los dispositivos finales tendrá un tamaño máximo de 128 Bytes.

Los nodos de la WPAN, coordinador y dispositivos finales, se etiquetaron con el estereotipo `<<GaExecHost>>` y `<<GaCommHost>>`. El estereotipo `<<GaExecHost>>` representan un procesador u otro dispositivo que ejecuta operaciones especificadas en el modelo, y hace las veces de host sobre el cual se ejecutan los `Step`.
El estereotipo `<<GaCommHost>>` representa el medio de comunicación entre el coordinador y los dispositivos finales. Como la tecnología utilizada para la WPAN es ZigBee [ZigBee:2006], el canal de comunicación, por limitaciones físicas del estándar [IEEE802.15:2006], tiene una capacidad de $250Kb/s$. La latencia de la WPAN es típicamente del orden de los $20ms$.
Los dispositivos finales y el coordinador tienen propiedades de sobrecarga de transmisión y recepción de mensajes, dadas en ms/KB. Los sensores y actuadores están etiquetados con el estereotipo `<<HW_Sensor>>` y `<<HW_Actuator>>`, y conectados a los dispositivos finales directamente en su backend a una velocidad de transmisión de $200Mb/s$.
El diagrama de actividades (véase Figura 3.20) detalla mucho más los requisitos de tiempo y recursos que necesita la WPAN para cumplir su función.

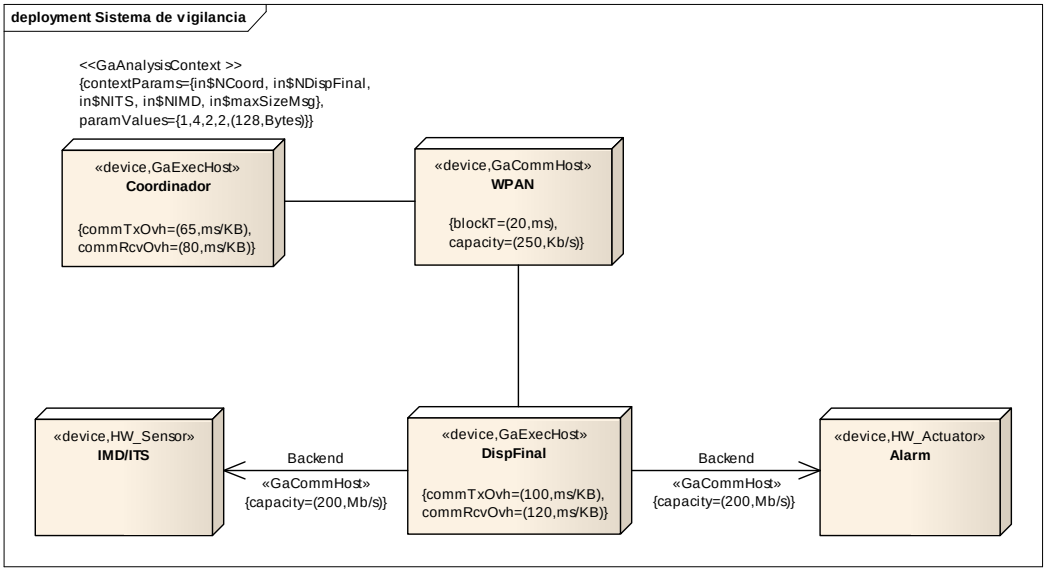

Figura 3.19: Diagrama de despliegue de la WPAN utilizando MARTE

Existe un `GaWorkloadEvent`, el cual garantiza el sensado de todos los dispositivos finales de la WPAN cada segundo, ya que dispara un nuevo evento inciador de sensado de un sector en forma periódica cada 300*ms*. `GaWorkloadEvent` sólo produce el evento disparador del sensado del sector, por lo tanto, las actividades que deben ejecutarse, antes de comenzar el sensado del siguiente sector, no deben tardar más que el periodo del `GaWorkloadEvent`.
Las actividades en este diagrama están estereotipadas con `GaScenario`, de esta forma, es posible asociarles atributos definidos en MARTE, y que son útiles para analizar y modelar desempeño. Los escenarios resaltados con rojo, son aquellos que se ejecutarán periódicamente mientras no se detecte un intruso en uno de los sectores del campamento militar. Estos escenarios tienen un `respT`, definido en la Tabla 2.2 de la sección 2.5.1.5, que permiten verificar el cumplimiento del periodo de muestreo de la red. Nótese que el escenario `getValueArea`, también tiene asociado el atributo `throughput`, el cual garantiza que este escenario debe ejecutarse por lo menos 3 veces por segundo, una vez para cada sector.
El tiempo de respuesta del escenario `getValueArea` incluye el tiempo de respuesta del escenario `getValueSensor`, y el tiempo que tarda la comunicación en viajar entre el coordinador y el dispositivo final y viceversa. `getValueArea` durante su ejecución envía un mensaje de solicitud a `getValueSensor`, y sólo hasta que este último responde, `getValueArea` termina su actividad. Por lo tanto, dentro de los 150*ms* de tiempo de respuesta del escenario `getValueArea` están los 60*ms* de tiempo de respuesta del escenario `getValueSensor`. De esta forma, `getValueArea` debe tener un tiempo de respuesta máximo de 90*ms*, incluído el tiempo de comunicación y latencia de la WPAN. Si no se detectan intrusos en un área, los tres escenarios en fondo rojo, que componen el proceso de sensado de un sector de la WPAN, no deben en conjunto tener un tiempo de respuesta mayor a 300*ms*, que es el tiempo entre inicio e inicio del escenario `getValueArea`. Para este caso, el tiempo de respuesta es 160*ms*. En el peor de los casos se detectará intrusos, y por lo tanto

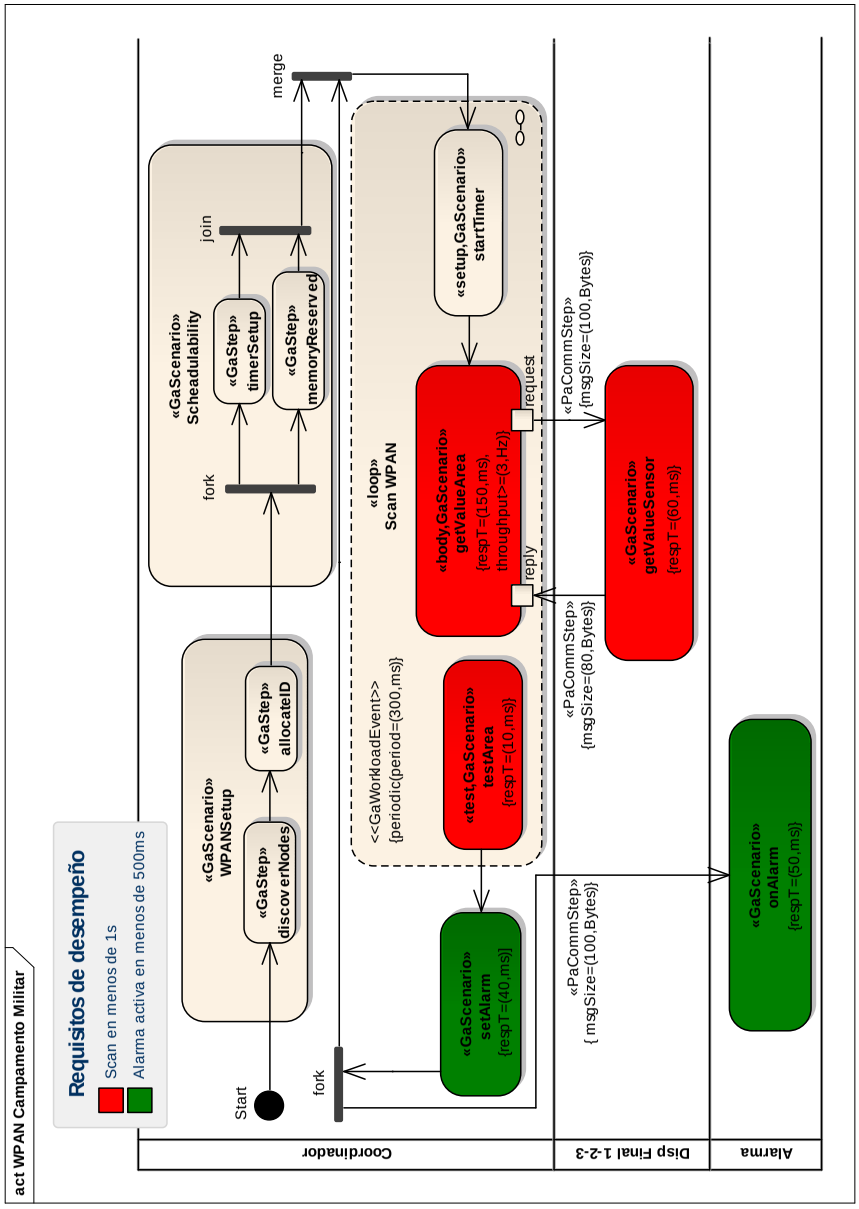

Figura 3.20: Diagrama de actividades de la WPAN utilizando MARTE

el tiempo de respuesta máximo pasará a ser 200*ms*, ya que entra en juego el escenario `setAlarm`, tiempo que sigue siendo menor que el requerido por cada evento del `GaWorkloadEvent`.
El NFR que exige que la alarma se dispare en menos de 500*ms* luego de detectarse un intruso, incluye los escenarios en rojo y en verde. Estos escenarios se ejecutan en 250*ms*, lo que sigue cumpliendo la exigencia del cliente. Sin embargo, aquí no está incluído el tiempo que le toma al mensaje enviado por `setAlarm` en llegar al escenario `onAlarm`. Ese tiempo de viaje se puede calcular como el tamaño del mensaje multiplicado por los atributos `commTxOvh` o `commRcvOvh`.

El diagrama de secuencias que se obtiene a partir del diagrama de actividades y el diagrama de clases (que es el mismo que se representa en la Figura 3.13), puede verse en la Figura 3.21. Hacer cumplir los NFRs temporales especificados por el cliente en el diagrama de actividades, implica que a los pasos en los que está dividida cada actividad, que puede ser una secuencia de métodos, se les asigne diferentes requisitos en cuanto a tiempos de ejecución. En este caso, los dos métodos de la clase `BuilderFrame` se estereotiparon como `GaStep`, lo que permite asignar un tiempo de respuesta a cada uno, `respT`, e inclusive diferentes tiempos de respuesta para el mismo método. Esta diferencia de tiempos para el mismo método puede ser por dos razones:

- El mismo método se ejecuta en hosts diferentes. Debido a los recursos, la capacidad y la carga de trabajo del host, es deseable establecer un tiempo de ejecución para cada situación. Esto permitirá maximizar el uso de los recursos del sistema
- El mismo método hace parte de escenarios diferentes. En este caso el método puede ejecutarse o no en el mismo host. Si es en el mismo host, puede ser que el escenario del cual hace parte el método, tenga unos requisitos de tiempo diferentes a otro escenario que también utilice el método. Esto puede deberse al hecho de que se establezcan diferentes niveles de importancia a cada escenario dentro del mismo host. En este caso, si el método participa en n escenarios, el tiempo de respuesta con el cual se debe construir el método, debe corresponder con el mínimo de esos tiempos de respuesta (véase ecuación 3.1), esto es:

$$respT = min(rspT_0, respT_1, ..., respT_{n-1}) \tag{3.1}$$

La frecuencia de sensado de los sectores, 3 por segundo, se representa por un `GaWorkloadEvent` asociado al fragmento estructurado `scan` de tipo `loop`.

Trasladar requisitos temporales sobre los métodos del diagrama de secuencias al diagrama de clases utilizando MARTE y VSL no es posible, ya que el contexto serian las clases, y no es sobre estas que queremos aplicar las NFPs, como `rspT`. La única forma sería por medio de notas asociadas a una clase, e indicando que la restricción se aplica a un método en particular de esa clase. Todo esto en lenguaje natural, o por lo menos una combinación con VSL.

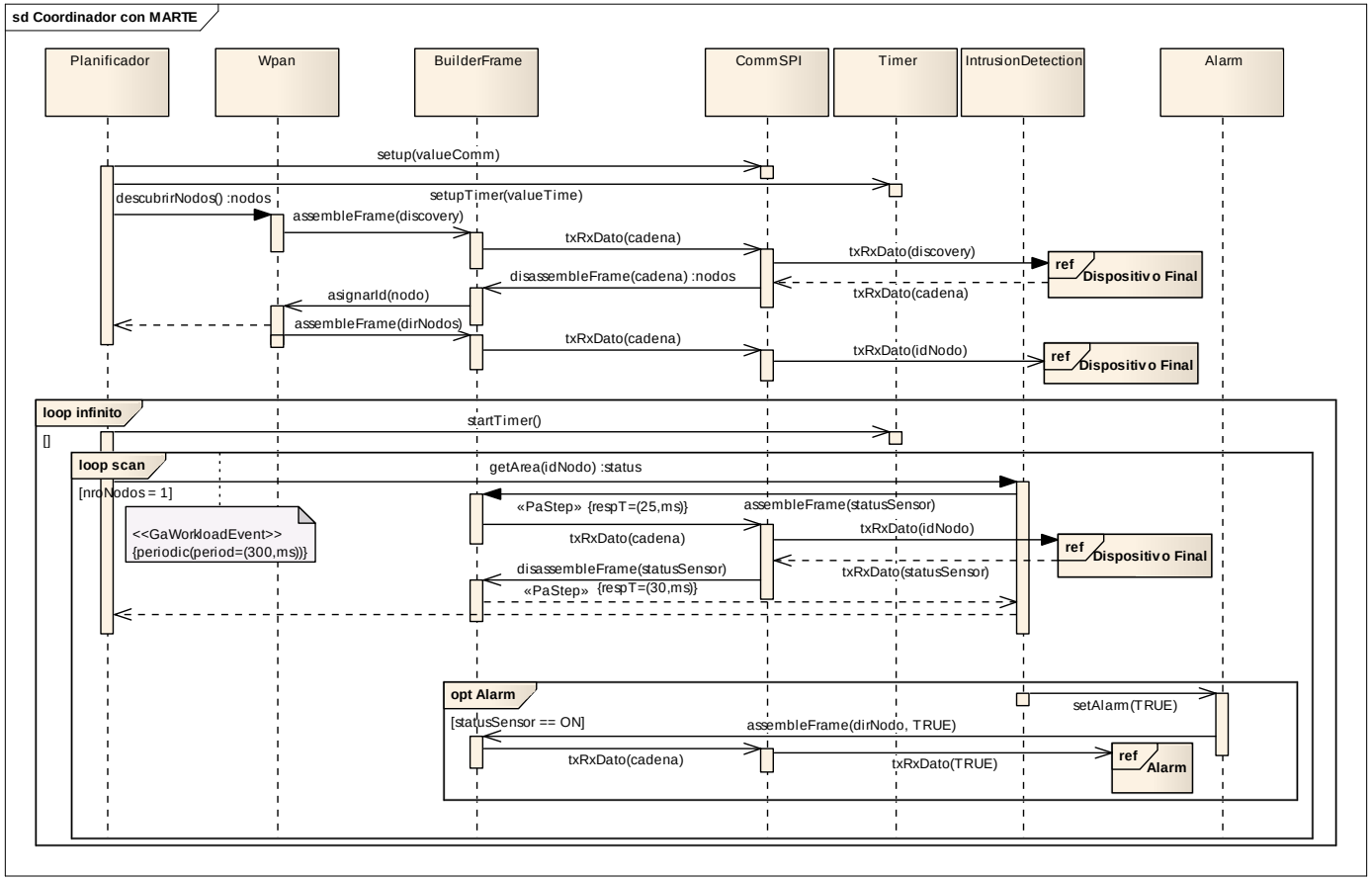

Figura 3.21: Diagrama de secuencia del Coordinador de la WPAN utilizando MARTE

La representación de NFRs con MARTE nos permite trabajar con básicamente cualquier NFP temporal y de espacio, con las cuales se pueden construir NFRs de desempeño de cualquier clase. Estas especificaciones se representan fundamentalmente en diagramas de comportamiento, y algunas es posible presentarlas en algún diagrama estructural (como es el diagrama de despliegue). Sin embargo, al momento de la implementación de las clases del sistema, las NFPs no pueden fácilmente representarse en, quizás, el diagrama más importante de cualquier modelo representado con UML, el diagrama de clases. Y esto se debe fundamentalmente al hecho de que no existe un soporte semántico para ello, ni una sintáxis definida que faciliten esta importante especificación para el desarrollador de las clases.

3.2.3.4. Solución: NFRs con UML

En [Tonu:2006] [Tonu:2005] se propone un marco de trabajo que incorpora NFRs dentro de una notación estándar de UML sin proponer extensión alguna. El marco de trabajo consta de tres procesos principales: la identificación de RFs y NFRs, la especificación de RFs y NFRs, y por último la integración de los NFRs con los FRs.
El proceso de identificación parte de diagramas UML y técnicas de educción de requisitos. La especificación se realiza según el marco de trabajo basado en softgoal [Chung:2000] y una plantilla propuesta por los autores, los cuales son almacenados en un repositorio para posterior reutilización. La integración de los NFRs con los FRs se puede lograr presentando estos primeros como componentes adicionales de XMI.
Para el sistema de vigilancia de la WPAN, los tres procesos definidos en este trabajo dan como resultado lo siguiente: el NFR identificado será el tiempo de scan de cada nodo en menos de 300*ms*, y la especificación en el marco de trabajo basado en softgoal [Chung:2000] se puede ver en la Figura 3.4. En este punto se requiere identificar métodos y atributos que se puedan asociar con el NFR tiempo de respuesta (véase Figura 3.13). Para nuestro caso tomaremos el método `assembleFrame` de la clase `BuilderFrame` y el método `txRxDato` y el atributo `capacity` de la clase `CommSPI`, para construir la nueva clase `TiempoDeRespuesta`, la cual se relaciona con las clases padres por medio de una asociación uno a uno (véase Figura 3.22).

En resumen, el proceso de incorporación de los NFRs en el diagrama de clases consistió en:

- Identificar métodos y atributos que contribuyan significativamente al NFR tiempo de respuesta: métodos `assembleFrame` y `txRxDato`, y el atributo `capacity`.
- Construir una nueva clase, de nombre TiempoDeRespuesta, con los métodos y atributos identificados en el paso anterior
- Relacionar esta nueva clase con las clases padres, en una relación uno a uno
- Borrar los métodos y atributos que conforman la nueva clase de las clases originales

En [Cysneiros:2001] [Cysneiros:2001a] [Cysneiros:2004] se propone una estrategía para elicitar NFRs e integrarlos en modelos conceptuales expresados en UML. Se muestra como integrar NFRs a los diagramas de clase, secuencia y colaboración, así como adaptar los casos de uso y los escenarios

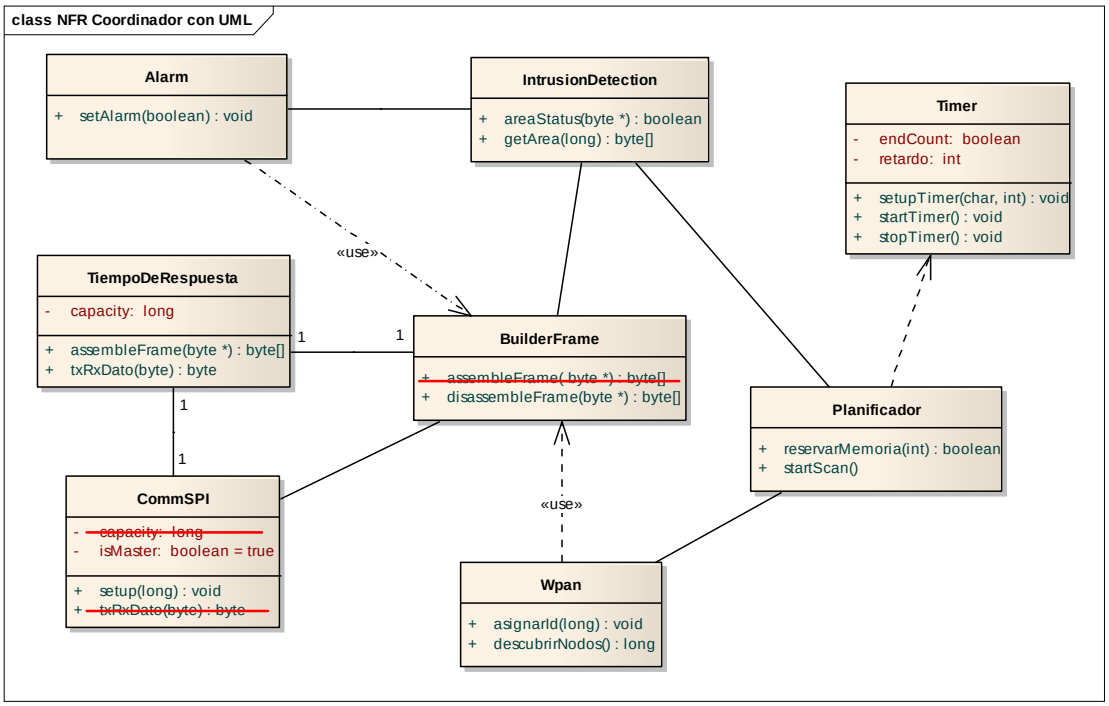

Figura 3.22: Diagrama de clases para representar el NFR tiempo de respuesta

para tratar con dichos NFRs.

Los autores identifican dos perspectivas diferentes en el proceso de desarrollo de software, la que tiene que ver con los aspectos funcionales y la que tiene que ver con los no funcionales. Se introduce el uso de LEL [Leite:1993] para nombrar las clases, y se detalla la forma para crear la perspectiva no funcional y su integración con la funcional. LEL juega un rol clave para la educción de requisitos, la cual se enfoca en el marco de trabajo basado en NFRs [Chung:2000]. Por último, se establecen unas heurísticas para identificar conflictos entre NFRs y la forma como tratarlos. A pesar de la posibilidad de trabajar con múltiples artefactos UML, nos centraremos exclusivamente en aquellos que utilizan el diagrama de clases para manejar los NFRs.

Con LEL se construye el vocabulario del universo del discurso, lo que permite una integración de las vistas funcional y no funcional. El principal objetivo de LEL es registrar palabras o frases peculiares al campo específico de aplicación. LEL inicialmente está compuesto de símbolos relacionados con los FRs, lo que exige extenderlo con el fin de consignar información relacionada con los NFRs.

La integración se basa en el uso de LEL como un ancla entre el SIG y el diagrama de clases, esto significa que cada NFR en el SIG se refiere a un símbolo LEL, y cada clase del diagrama de clases tiene que nombrarse usando un símbolo de LEL. De esta forma, el proceso de integración se centra en la búsqueda de un símbolo que aparezca en ambos modelos y posteriormente evaluar el impacto de adicionar el NFR operacionalizable al diagrama de clases. Por ejemplo, si consideramos el NFR tiempo de respuesta del Coordinador, representado en la Figura 3.4 y detallado en la Figura 3.23, podemos considerar los softgoal operacionalizables dinámicos como operaciones de la nueva clase

`TiempoDeRespuesta`, estereotipada como `<<NFR>>`; mientras que los softgoal estáticos pasan a ser atributos de la nueva clase (véase Figura 3.24).

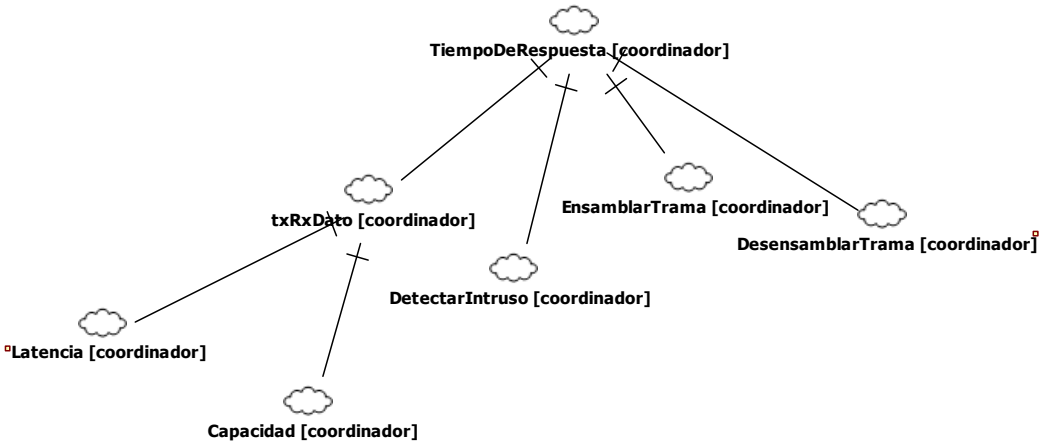

Figura 3.23: SIG detallado para el NFR tiempo de respuesta del Coordinador

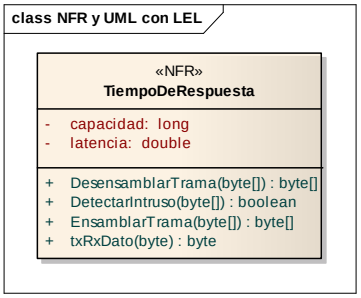

Figura 3.24: Integración del NFR tiempo de respuesta y el diagrama de clases

En ambos métodos se hace un acercamiento real de los NFRs al diagrama de clases. Se propone una integración de los FRs con los NFRs en la fase de diseño. Ambos métodos exigen el uso de heuristicas que le quitan rigurosidad, ya que vuelven la creación de las clases asociadas a los NFRs una decisión de diseño. En ninguna de las dos propuestas se logra caracterizar en forma cuantitativa el tiempo de ejecución de los métodos ni ninguna otra métrica temporal, lo que no permite especificar propiedades de desempeño del sistema

3.2.4. Solución: NFRs con ontologias

Ya que el uso de ontologias del dominio se ha extendido para soportar los procesos de captura de requisitos software [Kaiya:2005] [Kaiya:2006], en [Jingbai:2008] se propone un metamodelo, basado en una ontología del dominio, para NFRs de acuerdo al contexto de aplicación. Definir una ontología, en un lenguaje estándar como OWL, para describir los NFRs permite compartir y reutilizar el conocimiento, poder establecer reglas de inferencia lógica, y facilitar a las máquinas la lectura de la información.
Se propone un modelo dinámico del dominio, el cual se divide en dos niveles, procesos y objetivos, con sus respectivas asociaciones semánticas. Se utilizan ontologías para describir los modelos del dominio, y analizar la exactitud, consistencia y otras características de los requisitos software por medio de razonamientos ontológicos.
El metamodelo simplificado para los NFRs teniendo en cuenta el conocimiento del contexto puede verse en la Figura 3.25. La metaclase raiz es `Goal`, la cual sirve para representar requisitos a diferentes niveles, esto es, FRs y NFRs (`NFGoal`), los cuales pueden ser descompuestos y refinados por reglas definidas en el modelo del dominio. Cada objetivo de un sistema software, puede relacionarse con varios NFRs, de esta forma, cada `NFGoal` puede estar asociado con cero o varias declaraciones, `NFStatement`. De igual manera están definidos los tipos, valores y operaciones para los NFRs como se vio en la sección 2.2.2.

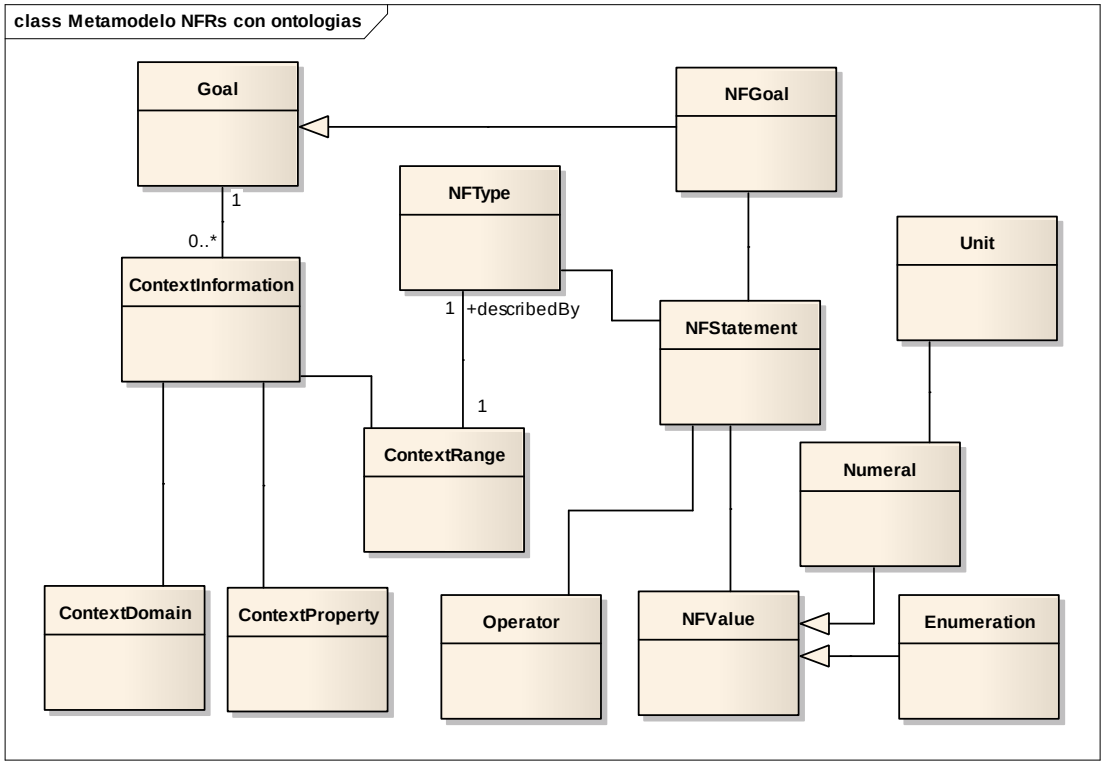

Figura 3.25: Metamodelo para los NFRs con conocimento del dominio (adaptado de [Jingbai:2008])

En la Figura 3.25 existe una relación `describeBy` entre `ContextType` y `NFType`. De esta manera la información del contexto se trata como una restricción de `NFGoal`. Siguiendo la relación definida en la ontología del dominio, el contenido del contexto puede ser transformado en un `NFValue`, lo que permite verificar la consistencia de los requisitos del usuario, así como descubrir aquellos requisitos ignorados.
Para realizar el proceso de educción de los requisitos de la WPAN, consideremos la seguridad del campamento como NFR, el cual tiene declaraciones de seguridad de la información, tiempo de respuesta y rendimiento, estos dos últimos hacen parte del tipo de NFR desempeño. Los valores, unidades y operaciones para las declaraciones están igualmente dados de acuerdo a las exigiencias de los usuarios: rendimiento mayor a $3Hz$, tiempo de respuesta menor a $300ms$ y seguridad de la información menor a la promedio (véase Figura 3.26).

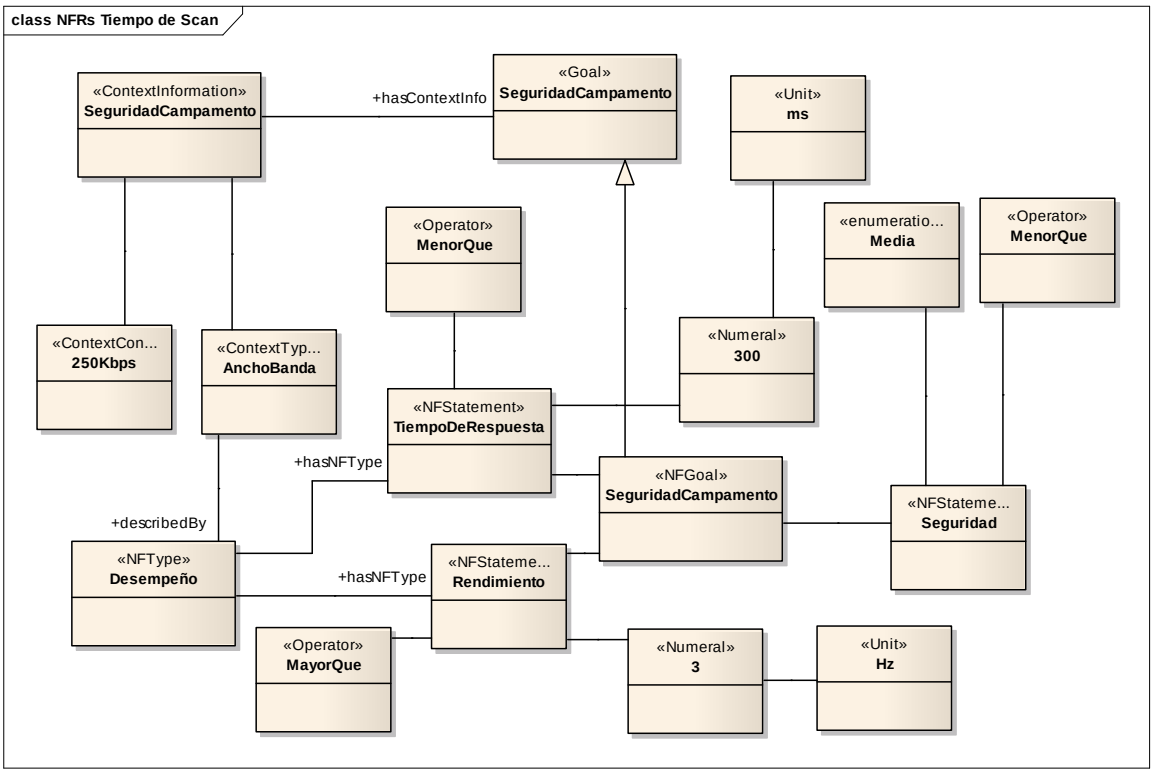

Figura 3.26: Instancia de NFRs basado en una ontología del dominio para la WPAN

Una ontología del dominio es útil para guiar el proceso de educción de NFRs. El modelo propuesto, según ODM [ODM:2009], soporta la descripción de NFRs con OWL-DL. El metamodelo para NFRs basado en ontologías del dominio, establece la relación entre el contexto y los NFRs de tal forma que se facilita el análisis sistemático de NFRs.
Sin embargo, esta metodología basada en ontologías del dominio se concentra fundamental-

mente en el proceso de educción de NFRs, no provee mecanismos para la representación de NFRs en etapas más adelantadas del proceso de desarrollo software.

3.2.5. Solución: Aspectos

En [Wehrmeister:2007] se presenta un marco de trabajo basado en aspectos para sistemas embebidos, distribuidos y de tiempo real, DERAF. DERAF provee un conjunto de aspectos, los cuales pueden extenderse, para tratar con NFRs de DERTS, a un nivel de abstracción muy alto, esto es, un conjunto de aspectos independientes de la implementación, que permiten manejara los NFRs durante la creación de modelos UML-RT durante la fase de diseño. Entre los NFRs que inicialmente se pueden manejar en DERAF están aquellos de desempeño, temporización, precisión, entre otros. A pesar del alto nivel de abstracción de los aspectos logrado con DERAF, la implementación de estos debe ser posible, evitando aspectos inviables.
Un conjunto reducido de aspectos de DERAF puede verse en la Figura 3.27. Con estos aspectos se adaptan los elementos del sistema modelado con el fin de manejar adecuadamente los NFRs, sin ligar los modelos DERTS a una solución específica. Abstraer los detalles relacionados con la implementación de los aspectos, asegura que la solución es independiente de la implementación seleccionada. Esto significa que el diseñador selecciona aspectos de acuerdo a cómo estos mejoran los elementos del sistemas, y define aquellos elementos que serán afectados por los aspectos seleccionados. Es de anotar que cada NFR puede ser manejado por varios aspectos.

Luego de modelados, los aspectos deben ser implementados, lo que exige que deba definirse el cómo y el dónde cada aspecto afecta los elementos del sistema.
La forma como se aborda el proceso es inicialmente realizar el análisis de requisitos según la metodología FRIDA [Bertagnolli:2003], la cual se adaptada al dominio de los sistemas embebidos de tiempo real. Luego de la educción de los FRs y los NFRs por medio de aspectos, se debe llegar a un diagrama de casos de usos que especifica la funcionalidad del sistema y sus NFRs transversales.
Para el caso de estudio que nos ocupa, la WPAN, tomaremos el diagrama de casos de uso del Coordinador (véase Figura 3.9), en este se pueden identificar varios casos de uso que están relacionados con el NFRs tiempo de scan de un nodo. Los casos de uso identificados son: Ensamblar y Desensamblar Tramas, Transmitir y Recibir Mensajes y Solicitar Estado de los Sectores. Los dos primeros casos de uso tienen exigencias de tiempo, ya que la duración de ellos determinará si el sensado de cada nodo se realiza en menos de 300*ms*. El caso de uso Solicitar Estado de los Sectores se asocia al aspecto de desempeño, ya que este debe garantizar el rendimiento del scan, esto es, tres veces por segundo (véase Figura 3.28).

Los elementos del modelo serán afectados por los aspectos seleccionados para soportar los NFRs de acuerdo a los JPDD [Schauerhuber:2006]. Los JPDD se utilizan para describir la selección de los join point e indicar en que lugar del modelo se realizará la adaptación [Stein:2006]. Los JPDD pueden ser modelados utilizando alguno de los diagramas de comportamiento de UML, esto es, diagrama de estados, de actividades, de secuencia, entre otros.
Luego de identificar los join point y los puntos de adaptación, por ejemplo en el diagrama de se-

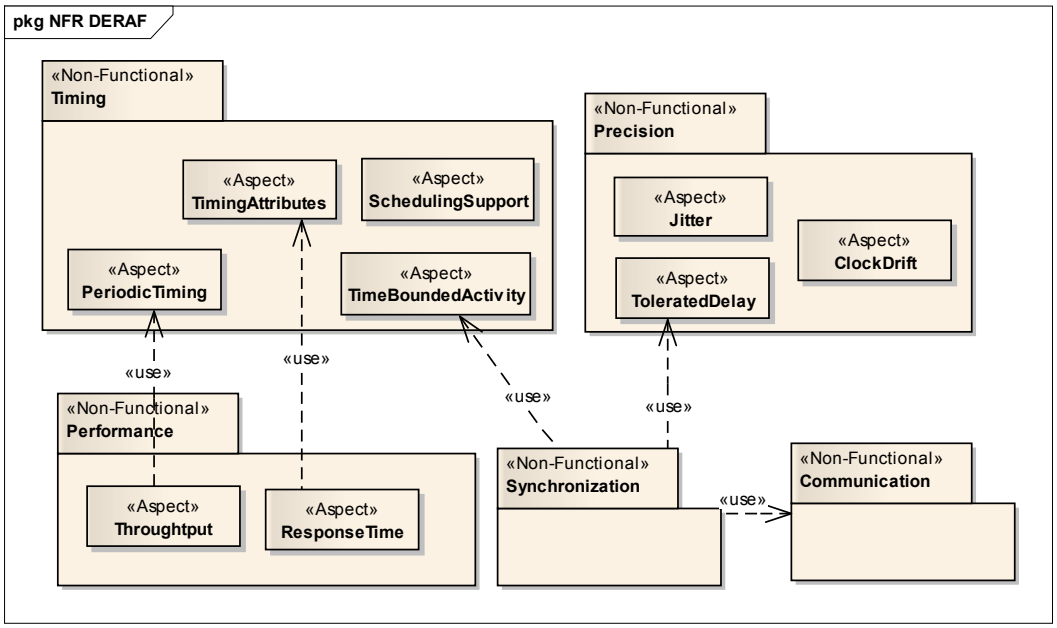

Figura 3.27: Conjunto de aspectos de DERAF para el manejo de NFRs (adaptado de [Wehrmeister:2007])

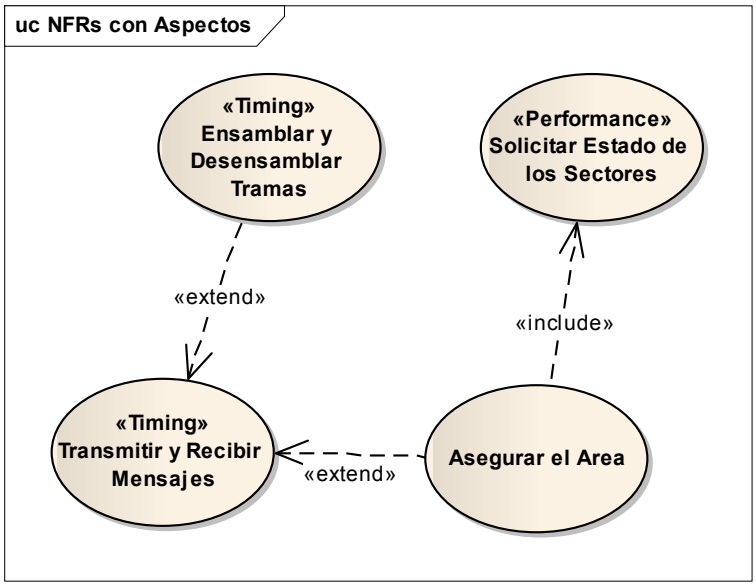

Figura 3.28: Casos de uso para la WPAN con estereotipos de NFRs

cuencias de la Figura 3.14, se continua con la representación de los aspectos como una clase con el estereotipo `<<Aspect>>`. Los join point se representan como atributos, mientras que las adaptaciones vienen siendo las operaciones. Los joints points identificados para el aspecto Throughput, que aporta al NFR Solicitar Estado de los Sectores, tiene dos posibles atributos, su frecuencia y el número de repeticiones. Las operaciones son estructurales o dinámicas. Para las estructurales el punto de aplicación de la adaptación siempre es el mismo, para las dinámicas puede cambiar. `Setup` siempre se aplicará al principio, para las otras dos operaciones puede variar su punto de inserción. La naturaleza transversal de los NFRs hace que los aspectos afecten a varias clases del modelo UML del sistema. En este caso las clases afectadas por el aspecto Throughput son: `Timer`, `Planificador` e `IntrusionDetection`. Esta transversalidad se representa con la asociación estereotipada como `<<crosscut>>` (véase Figura 3.29).

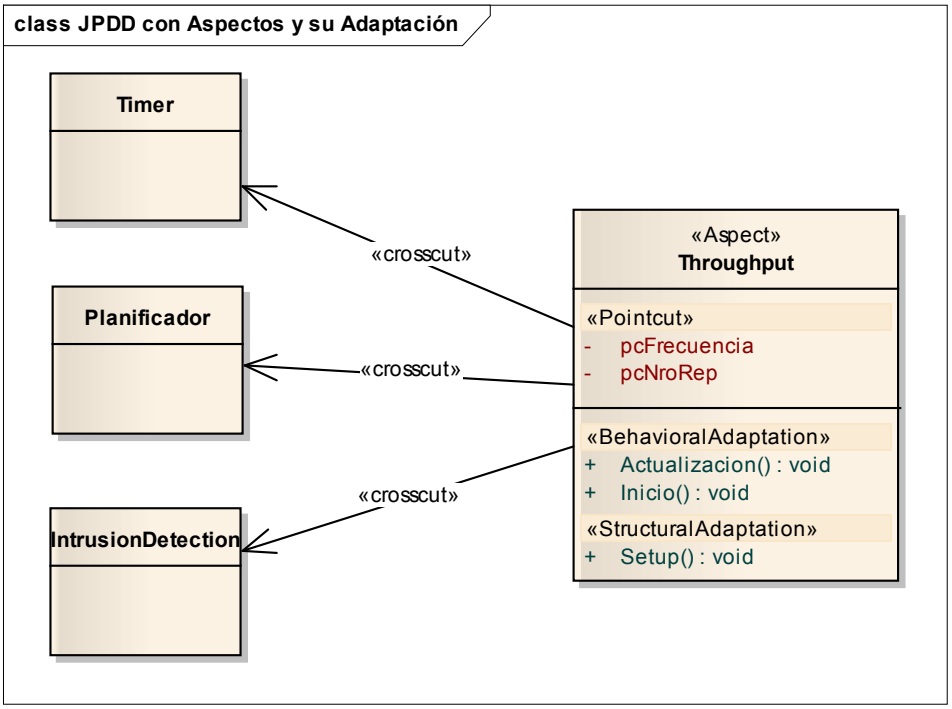

Figura 3.29: Vinculación de los JPDD con un aspecto y su adaptación

En [Dai:2005] [Dai:2006] se define el FDAF, un enfoque orientado a aspectos, con el fin de soportar el diseño y análisis de NFRs para sistemas distribuidos de tiempo real, usando UML y métodos formales existentes. En el FDAF, los NFRs se definen como aspectos reutilizables, definidos con UML.

Con el fin de soportar el diseño de estos aspectos y su posterior traducción a métodos formales, se hace necesario extender UML. Esta extensión consiste en la representación de los aspectos como subsistemas de UML, los cuales están definidos usando diagramas de clases, diagramas de secuencia, diagramas de colaboración u OCL.

FDAF contiene una herramienta que realiza la traducción, garantizando la consistencia, del diseño semiformal UML extendido al modelo formal seleccionado. ¿Qué método formal utilizar para cada aspecto del sistema? [Luckham:1995] [RAPIDE:1997] [RAPIDE:1997a] [RAPIDE:1997b] [Inverardi:1995] [Shaw:1995] [Allen:1997] [Wang:1999], esta es una decisión muy importante, ya que cada notación es útil para sólo una o unas pocas propiedades del sistema. FDAF asiste al usuario en la toma de esta decisión.

Debe tenerse en cuenta que un modelo enfocado sólo en un tipo de propiedad, tiempo de respuesta por ejemplo, es generalmente mucho más simple y pequeño que aquel que incluye múltiples propiedades. De esta forma, un conjunto de modelos simples, cada uno diseñado para un aspecto específico del sistema, pueden construirse y evolucionar en forma relativamente independiente. Esto facilita el modelado y análisis del sistema. Una vez traducido al modelo formal, este puede analizarse para aspectos específicos utilizando herramientas existentes.

El proceso de modelado con el FDAF consiste de tres actividades: la creación del modelo de diseño orientado a aspectos según la extensión de UML propuesta, la creación del modelo de diseño formal orientado a aspectos y el análisis del modelo de diseño orientado a aspectos. La primera actividad es iterativa y cada vez refinada con el fin de crear la línea base de la arquitectura. Luego de la tercera actividad se debe analizar el resultado y verificar si los NFRs si se satisfacen, si no, debe revisarse el modelo UML original.

Para el caso de la WPAN, tomaremos el diagrama de estados de la Figura 3.18. En este diagrama, el tiempo de respuesta del estado `checkSectores` del planificador no debe ser mayor a 333*ms*, lo cual se representa en la Figura 3.30. Nótese que a los dos estados representados en el diagrama se adicionó un paralelogramo como notación para el aspecto que soporta la propiedad tiempo de respuesta. En este caso el aspecto se estereotipo con el elemento del perfil MARTE `GaStep` [MARTE:2009], el cual representa un paso en un escenario de análisis de desempeño. La propiedad `respT` representa el tiempo de respuesta del paso, que en este caso asumimos que será máximo de 333*ms*.

En [Jain:1991] se sugieren varias técnicas para la evaluación del desempeño: medición, modelamiento analítico y simulación. UML no tiene tales herramientas de análisis, por lo tanto se utiliza el modelo UML extendido que incluye el aspecto para el tiempo de respuesta, y se traduce en un modelo formal Rapide. Esto lo realiza automáticamente la herramienta case del FDAF, arrojando una cláusula de tiempo Rapide. La simulación se realiza utilizando una herramienta como RPDC, POV o RAPTOR.

Por falta de disponibilidad de las herramientas, no se realizó la traducción automática a Rapide ni el análisis del modelo formal obtenido.

En ambos métodos se aprovecha la transversalidad de los NFRs para ser representados con aspectos, de esta manera se puede concentrar el NFR en un sólo componente del modelo, y no dispersarlo por todo el diagrama. En [Wehrmeister:2007] se obtiene un modelo en el que se asocia el aspecto, que representa al NFR, con las clases que afecta, pero no establecen la forma como llevar a cabo la implementación que satisfaga el NFR.

La metodología propuesta en [Dai:2005] requiere conocimientos en métodos formales, y no

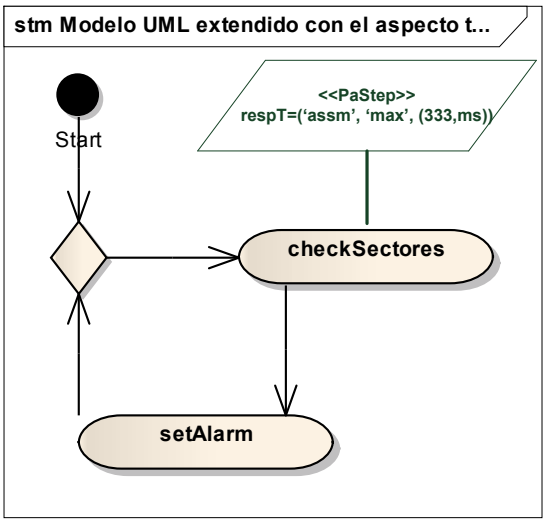

Figura 3.30: Diagrama de estados del modelo UML extendido con el aspecto tiempo de respuesta

sólo en uno, lo que dificultad su implementación y análisis, ya que estos métodos son verdaderamente más difíciles de dominar. Esta propuesta se enfoca en las etapas de análisis y modelado, y no en la implementación de la solución.

3.2.6. Solución: XML

En [Fei:2007] se presenta un método, basado en XML, para modelar NFRs. Se definen tres abstracciones: NF-Attribute, NF-Constraint y NF-Operation. NF-Attributes modela características o atributos no funcionales en forma descriptiva, no permite que estos sean cuantificados. NF-Constraint modela las restricciones sobre los atributos. Estas restricciones imponen condiciones sobre lo que es requerido en la implementación. De esta forma, se definen un conjunto de operaciones, NF-Operations, que deben implementarse con el fin de satisfacer los NF-Constraint. Las operaciones no funcionales modelan elementos software o hardware que afecten el atributo no funcional que se está restringiendo.
La estructura del modelo de datos para los NFRs utilizando XML se puede ver en la Figura 3.31, allí se identifican datos de uso común, como el ID, el nombre, el padre, el hijo y la prioridad. Aquellos que no son obvios son: Contribution de Attribute, KindOfEffect y AffectAttr de Operation, y KindOfConstraint.

Contribution indica el tipo de contribución que el atributo hijo realiza al atributo padre. Son cuatro las posibles contribuciones: All, Any, One y None.
AffectAttr define cuál atributo será afectado con la implementación de la operación. KindOfAffect define el efecto sobre el atributo debido a la implementación de la operación. Hay cinco posibles

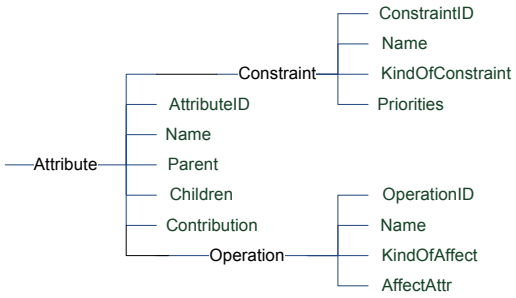

Figura 3.31: Estructura del modelo de datos para los NFRs [Fei:2007]

efectos: Affect+3, Affect+2, Affect+1, Affect 0 y Affect-1.
KindOfConstraint indica el efecto sobre el atributo que tiene la restricción. Hay cuatro posibilidades: Weak, Light, Strong y Present.
Considerando la descomposición de NFRs según la Figura 2.1, en la cual se destaca el tiempo como un subtipo del NFR desempeño, se tomarán los atributos tiempo de respuesta y rendimiento para modelar los requisitos de scan de la WPAN (frecuencia de scan de la red, 3 nodos por segundo, tiempo de scan de un nodo, menor a 333*ms*).
La definición del desempeño utilizando XML para los NFRs es:

```
<Attribute Contribution="All">
  <AttributeID> NFR01 </AttributeID>
  <Name> Tiempo </Name>
  <Parent> Desempeño </Parent>
  <Children> Rendimiento </Children>
  <Children> Tiempo de Ejecución</Children>
  <Constraint KindOfConstraint="Strong" Priorities="High">
    <ConstraintID> Const01 </ConstraintID>
    <Name> Buen Desempeño </Name>
  </Constraint>
  <Operation KindOfAffect="Affect+3">
    <OperationID> Op01 </OperationID>
    <Name> Frecuencia de Scan </Name>
    <AffectAttr> Rendimiento </AffectAttr>
  </Operation>
  <Operation KindOfAffect="Affect+2"></Operation>
    <OperationID> Op02 </OperationID>
    <Name> Tiempo por Scan </Name>
    <AffectAttr> Tiempo de Ejecución </AffectAttr>
  </Operation>
</Attribute>
```

Debe tenerse en cuenta que este modelado es sólo descriptivo, por lo tanto el caracter cuantitativo de los NFRs de la WPAN no puede representarse. Se tiene un sólo NFR, Tiempo, el cual tiene como padre a Desempeño e hijos a Rendimiento y Tiempo de Ejecución. La restricción lo que exige es Buen Desempeño, el cual se logra con las operaciones Frecuencia de Scan y Tiempo por Scan, que afectan a los atributos Rendimiento y Tiempo de Ejecución. Nótese que ambas operaciones tienen efectos favorables sobre los atributos, y por ende sobre el NFR. Se le asigna mayor efecto a Frecuencia de Scan que a Tiempo por Scan, ya que este último podría no ser necesariamente igual para todos los nodos, pero eso si dentro de un rango que no afecte la Frecuencia de Scan, que es al que mayor importancia se le asigna.
Este modelado está incluído dentro de un marco de trabajo propuesto por los autores. Este marco de trabajo incluye integración de NFRs con FRs a través de elementos de la arquitectura, como conectores, componentes, puertos e interfaces; una fase de refinamiento que incluya componentes, conectores, interfaces, NF-Attribute, NF-Constraint y NF-Operation; una fase de mapeo, que permite que los conectores, componentes y NF-Operation puedan corresponderse con elementos ejecutables; por último la fase de evaluación, la cual define si la arquitectura software debe o no ser rediseñada de acuerdo a si los NFRs se satisfacen o no.

Este enfoque tiene varias ventajas. Primero, el uso de un lenguaje tan simple como XML para modelar los NFRs. Segundo, la integración de los NFRs modelados con XML a la arquitectura software donde se incluye el diseño de los FRs. Tercero, la facilidad de extender este modelado utilizando XML Schema. Sin embargo, a pesar de las obvias ventajas existen limitantes en el modelado. Primero, el carácter descriptivo de la representación de los NFRs impide validar el cumplimiento de estos, ya que no es posible representar NFRs cuantitativos. Segundo, no existe una integración de XML a lenguajes y metodologías de desarrollo de software existentes.

3.3. Problemas al tratar con requisitos no funcionales

De acuerdo a lo revisado en este capítulo es posible identificar varios problemas al tratar con NFRs. Estos problemas se deben fundamentalmente a tres aspectos: la definición, la clasificación y la representación.

3.3.1. Problemas de definición

Son muchas las definiciones utilizadas para los NFRs, pero básicamente todas están construidas sobre los términos propiedad, característica, atributo, calidad, restricción y desempeño. Sin embargo, no existe un consenso acerca de lo que estos términos significan, ya que en muchas ocasiones son utilizados sin una definición previa o un ejemplo que lo clarifique. Por esta razón, se evidencian problemas conceptuales y de terminología entre las diferentes definiciones.
En [Jacobson:2000] los NFRs se definen como propiedades, las cuales incluyen a las restricciones:

"*Un NFR es un requisito que especifica propiedades del sistema, tal como restricciones de implementación o del entorno de despliegue, dependencias de la plataforma, mantenibilidad, extensibilidad, confiabilidad y desempeño. Estos requisitos especifican restricciones físicas sobre FRs*", mientras que en [Wieger:2003] y [Anton:1997] las restricciones no están incluídas dentro de las propiedades. En [Anton:1997] se define: "*Un NFR describe los aspectos no comportamentales de un sistema, capturando las propiedades y restricciones bajo las cuales debe operar el sistema*"; mientras en [Wieger:2003] se tiene que: "*Un NFR es una descripción de una propiedad o característica que el sistema software debe exhibir o una restricción que debe cumplir, diferentes a aquellas observables en su comportamiento*".

Acerca de las restricciones, no existe un consenso sobre su definición, en [IEEE830:1998] y [Jacobson:2000] el término se utiliza sólo para la fase de diseño y para indicar las restricciones físicas. [Robertson:1999] las restricciones son tratadas como propiedades no funcionales, diferentes a las restricciones de diseño, las cuales son consideradas como un concepto diferente a los FRs y NFRs.

En [IEEE830:1998] no se definen el término NFR, pero se incluyen en los atributos del sistema, los cuales no tienen en cuenta al desempeño ni las restricciones. [Davis:1993] también utiliza el término atributo del sistema, y cada NFR lo cataloga como tal.

Cualquie requisito puede ser tratado como requisitos de calidad, ya que según [ISO:9000] "*La calidad es el grado en el cual un conjunto de características cumple los requisitos*". De esta forma, cada requisito puede ser tratado como una restricción, porque así se restringe el espacio de potenciales soluciones que reunen los requisitos. Como atributos de calidad los trata [Burge:2002]: "*Los NFRs describen atributos del sistema que contribuyen a la calidad del producto como un todo*"; [Botella:2001] trabaja sobre el conjunto de estándares ISO/IEC sobre calidad del software [ISO/IEC:1999], en el cual el eje central es el modelo de calidad: "*Un modelo de calidad es definido por medio de características generales del software, las cuales son refinadas en subccaracterísticas hasta llegar a atributos software medibles. Los NFRs son requisitos de calidad que pueden ser definidos como restricciones sobre el modelo de calidad*". En todos los casos, el término calidad implica un conjunto reducido de atributos de calidad diferentes a los relacionados con la funcionalidad, esto es, usabilidad, confiabilidad, seguridad, entre otros.

El alcance de los NFRs es otro aspecto que se trata en las definiciones, y sobre el cual no hay un total acuerdo. [Burge:2002] [Chung:2000] y [Davis:1993] los definen como requisitos globales o atributos generales, lo que separa definitivamente los FRs de los NFRs. [Burge:2002] cita: "*Los NFRs no se relacionan con un componente específico del sistema, ya que su naturaleza los hace transversales... Los NFRs no pueden ser evaluados sin mirar el sistema como un todo*"; mientras que [Chung:2000] dice: "*... requisitos globales sobre el desarrollo, desempeño, confiabilidad, mantenibilidad, portabilidad, robustez y demás*"; por último, [Davis:1993]: "*Un NFR es un atributo general requerido por el sistema, el cual puede ser portabilidad, confiabilidad, eficiencia, entre otros*". Por otra parte, [Jacobson:2000] enfatiza que los NFRs son locales, para el caso de la mayoría de los requisitos de desempeño, y globales para otros casos como confiabilidad y seguridad.

En casi todas las definiciones los NFRs son requisitos del producto, excluyendo aquellos requisitos del proyecto, del proceso y del entorno de despliegue. Por ejemplo, en [Sousa:2004]: "*Los NFRs son requisitos que imponen restricciones sobre el producto que se está desarrollando*". Sin embargo, en [Kotonya:1998] se define que: "*Los NFRs son requisitos que no están específicamente relacionados*

con la funcionalidad del sistema. Imponen restricciones sobre el producto y el proceso de desarrollo, y además especifican restricciones externas que el producto debe reunir", esta definición considera a los requisitos del proceso como NFRs.

3.4. Problemas de clasificación

A parte de los problemas de definición, se encuentran igualmente bastantes problemas de clasificación de NFRs. [Davis:1993] utiliza el árbol de clasificación definido en [Boehm:1976], en el cual se tratan los requisitos de calidad como una subclasificación para los NFRs. En [IEEE830:1998] se realiza una subclasificación de NFRs en requisitos de interfaz, requisitos de desempeño, atributos y restricciones de diseño, donde estos atributos son un conjunto de aspectos de calidad, entre ellos están, confiabilidad, disponibilidad, seguridad, entre otros.
En [IEEE:1990] se distinguen FRs, y su contraparte, que se corresponden con los NFRs, son los requisitos de diseño, implementación, interfaz, desempeño y físicos.
En [Sommerville:2007] se clasifican los requisitos en aquellos que son del producto, de la organización y los externos.
La ISO [ISO/IEC:1999] (véase sección 2.2.2.1), define los NFRs como requisitos de calidad, los cuales categoriza como de funcionalidad, de confiabilidad, de usabilidad, de eficiencia, de mantenibilidad, y de portabilidad.
Con todo esto, es posible listar un conjunto de NFRs más específicos, que no necesariamente son más claro para el lector, entre estos están:

Accesibilidad, adaptabilidad, disponibilidad, capacidad, cohesión, tiempo de comunicación, costo y tiempo de integración de componentes, comprensibilidad, confidencialidad, controlabilidad, exactitud, fiabilidad, distributividad, extensibilidad, viabilidad, interoperabilidad, facilidad de aprendizaje, costo de mantenimiento, desempeño, modificabilidad, nomadicidad, precisión, productividad, prontitud, configurabilidad, reconfigurabilidad, reemplazabilidad, capacidad de respuesta, estabilidad, seguridad, simplicidad, capacidad de formación, puntualidad, comprensibilidad, usabilidad, variabilidad, visibilidad, aditividad, agilidad, claridad, compatibilidad, conceptualidad, costo, lealtad al cliente, descomposición, diversidad, eficiencia, evolucionabilidad, flexibilidad, orientación, modularidad, observabilidad, consistencia, fiabilidad, recuperabilidad, replicabilidad, escalabilidad, sostenibilidad, rendimiento, tolerancia, transferibilidad, garantia, facilidad de uso, verificabilidad, ajustabilidad, auditabilidad, capacidad, completitud, componibilidad, concisión, coordinación, acoplamiento, personalización, tiempo de desarrollo, elasticidad, costo de ejecución, tolerancia a fallos, formalidad, informatividad, integridad, intuitibidad, mantenibilidad, mobilidad, madurez, naturabilidad, costo de planificación, portabilidad, tiempo de prototipo, tiempo de respuesta, recuperabilidad, repetibilidad, reusabilidad, robustez, similaridad, especificidad, subjetividad, supervivencia, comprobabilidad, trazabilidad, transparencia, uniformidad, validez, versatilidad, entre otros.

Es clara la dificultad para clasificar los NFRs, ya que inclusive muchos de los términos son difíciles de entender, pero a pesar de ello, esta tesis se centra en los requisitos de desempeño, los cuales incluyen, entre otros, propiedades de tiempo y espacio.

3.5. Problemas de representación

Cuando los NFRs se refinan hasta el punto de operacionalizarse [Chung:2000], se puede pasar de un NFR a un FR. Por ejemplo, “El sistema no puede permitir accesos no autorizados a los datos del cliente", es definitivamente un requisito de seguridad, clasificado como NFR. Pero si este NFR se refina se puede llegar a, “La base de datos sólo garantizará acceso a aquellos clientes que tengan nombre de usuario y contraseña", que sigue siendo un requisito de seguridad, pero ahora es FR. A pesar de que ambas descripciones corresponden a un requisito de seguridad, las representaciones son completamente diferentes, ya que el tipo de requisito ha cambiado.
La documentación de los requisitos es otro problema de representación, ya que algunos autores sugieren que la documentación de los FRs y los NFRs se haga por separado [Robertson:1999] [IEEE830:1998], mientras que en metodologías de desarrollo basadas en casos de uso, como RUP por ejemplo, los NFRs se mantienen ligados a estos mientras sea posible [Jacobson:2000].

Capítulo 4
Planteamiento del Problema de Investigación

Resumen En este capítulo se justifica la especificación de NFRs desde su conveniencia, sus implicaciones teóricas y prácticas, y la utilidad metodológica que aporta a los procesos de educción de requisitos software. Igualmente, por medio de las preguntas de investigación, se plantea de forma directa y sin distorsión el problema a atacar. Finalmente, se establecen los objetivos de la tesis.

Los NFRs han generado amplios debates entre las personas que trabajan en RE, ya que no existe un consenso acerca de lo que estos son y cómo deben elicitarse, documentarse y validarse. Sin embargo, se tiene claro que los NFRs son de los requisitos más importantes que un sistema software debe satisfacer, ya que su no cumplimiento, generalmente trae consigo errores de muy alto costo en tiempo y dinero, e inclusive pueden comprometer el éxito del proyecto.
Los NFRs pueden ser subjetivos, ya que pueden verse, interpretarse y evaluarse en forma diferente por diferentes personas. La pobre especificación, resumida y vaga, hace este problema bastante complejo.
Los NFRs también pueden ser relativos, ya que la interpretación e importancia del NFR varia dependiendo del sistema que se este considerando. Por ejemplo, la seguridad es crítica para un sistema bancario, pero no tanto para un sistema de monitoreo de tempratura de una sala de conferencias. Así mismo, el requisito seguridad tiene una interpretación dependiendo del sistema. No es lo mismo la seguridad del sistema de control de un reactor, a la seguridad del sistema de información académica de una universidad.
La satisfacibilidad de un NFR también puede ser relativa, ya que esta depende de las circunstancias, no es lo mismo el tiempo de respuesta de un servidor en la noche, al tiempo de respuesta durante la hora pico. Por esta razón, una solución óptima bajo todo punto de vista, puede no ser alcanzable.
Otro aspecto fundamental de los NFRs es que interactúan entre ellos, lo que hará que un NFR pueda ayudar a que otro NFR se satisfaga o no. Por ejemplo (véase sección 3.2.2), los NFRs `TiempoDeRespuesta` y `Throughput` contribuyen a mejorar el desempeño de un sistema, mientras que el NFR `Seguridad[info]` va en la vía contraria, desmejora el desempeño del sistema.
Debido a que los NFRs tienen un impacto global sobre el sistema, debe tenerse cuidado a la hora de solucionar problemas presentados con estos, ya que la transversalidad de un NFR exige que múltiples clasificadores de un modelo deban tenerse en cuenta a la hora de evaluar su impacto.

Por ejemplo, para aumentar el nivel de seguridad de un sistema se hace necesario implementar algoritmos de encriptación que permitan firmar digitalmente un documento. Esto, indudablemente aumenta el tiempo de respuesta de ese clasificador, lo que afecta globalmente el desempeño del sistema.

4.1. Justificación

Son múltiples los aspectos que pueden justificar este trabajo, en este caso destacaremos cuatro: la relevancia, las implicaciones prácticas, el valor teórico y la utilidad metodológica.
La especificación formal de NFRs temporales no sólo es importante, también es necesaria, cuando se está realizando el proceso de educción de requisitos y modelado del sistema, ya que esto permite mejor la comunicación con los usuarios y entre los desarrolladores, con el fin de construir el sistema deseado por el cliente. Sin esta representación y formalización de NFRs temporales la comunicación pasa a ser ad hoc y no se puede garantizar el cumplimiento de los requisitos. Además, una verbalización del NFR temporal claramente dejará en el aire muchas ambigüedades.
La especificación formal de NFRs temporales soluciona un problema que actualmente se le presenta a los modeladores de sistemas de tiempo real, y es la representación de dichos NFRs en el diagrama de clases de UML. Esta representación en el diagrama estructural más importante de UML, permite eliminar la ambigüedad a la hora de comunicar a los desarrolladores las restricciones de tiempo del sistema, facilitando así la construcción del mismo, y garantizando que sus exigentes requisitos si se están cumpliendo.
Esta especificación de NFRs temporales en el diagrama de clases tiene un aporte teórico importante, y es que actualmente, según la especificación de UML [UMLSuper:2009], no es posible especificar propiedades a algunos artefactos UML, como las operaciones. Sin embargo, con esta propuesta se pretende formalizar la relación entre propiedades y operaciones, lo que abriría un mundo de posibilidades a la especificación de múltiples tipos de sistemas, no sólo los embebidos de tiempo real.
El aporte metodológico del trabajo está en la integración de la nueva especificación de NFRs temporales en metodologías de desarrollo software tradicionales, teniendo en cuenta que se puede incluir el proceso de educción de requisitos como el de modelado del sistema. Todo esto debe utilizarse en metodologías que consideren en la etapa de modelado a UML como su lenguaje.

4.2. Preguntas de investigación

Existen varios estilos de preguntas de investigación que pueden orientarnos a alcanzar los objetivos de esta tesis. En este trabajo, plantearemos tres tipos de preguntas: descriptivas, de relación y de diferencia.
Dentro de las preguntas de tipo descriptivas están:

- ¿Cuáles son las características fundamentales de los requisitos no funcionales?
- ¿Qué metodologías de desarrollo software se han utilizado para realizar los procesos de educción y representación de requisitos no funcionales?

- ¿Qué técnicas de formalización de requisitos funcionales y no funcionales existen? ¿Pueden estas técnicas utilizarse para la especificación de requisitos no funcionales?

Entre las preguntas de relación podemos plantear las siguientes:

- ¿Qué relación existen entre los requisitos funcionales y no funcionales?
- ¿Qué relación existe entre las técnicas de educción de requisitos funcionales y no funcionales?
- ¿Cómo están relacionados los requisitos no funcionales con las restricciones de tiempo de algunos sistemas?

Las preguntas de diferencia que pueden apoyarnos en el desarrollo de esta tesis son:

- ¿Qué sucede cuando en el proceso de educción de requisitos del sistema se incluyen tanto los requisitos funcionales como los no funcionales? ¿Es posible utilizar las mismas técnicas de educción para ambos tipos de requisitos?
- ¿El proceso de educción de requisitos no funcionales cambia cuando se consideran los requisitos temporales no funcionales?
- ¿La metodología de desarrollo software a emplear depende de qué tipo de requisitos se consideren? ¿Es posible utilizar el lenguaje de modelado unificado para especificar las características particulares de los requisitos no funcionales?

4.3. Objetivos

Este trabajo es de carácter teórico, conceptual y metodológico, ya que plantea una manera novedosa de especificar los NFRs temporales en el diagrama de clases de UML, esto implica extender el metamodelo del diagrama de clases y del diagrama de expresiones de VSL para abarcar propiedades de las operaciones actualmente no cubiertas, así como facilitar la definición de contextos del sistema en el diagrama de clases de UML de tal forma que la semántica no representa ninguna ambigüedad.

4.3.1. Objetivo general

Especificar formalmente en el diagrama de clases de UML los requisitos no funcionales temporales de un sistema.

4.3.2. Objetivos específicos

- Integrar el proceso de elicitación y especificación de requisitos temporales no funcionales en metodologías de desarrollo software existentes.
- Representar en el diagrama de clases de UML requisitos temporales no funcionales relacionados con las operaciones de una clase.

- Especificar formalmente en el diagrama de clases de UML requisitos temporales no funcionales asociados a operaciones de una clase.
- Validar la propuesta de especificación de requisitos temporales no funcionales en el diagrama de clases de UML con un caso de estudio.

Capítulo 5
Método Propuesto

Resumen Este capítulo presenta la propuesta de esta tesis. Inicialmente se enfoca en la metodología para la educción de requisitos, metodología que está basada en UN-METODO. Posteriormente, con el fin de especificar formalmente los requisitos no funcionales – NFRs en el diagrama de clases, se propone una modificación al metamodelo de este, y al metamodelo del diagrama de expresiones del lenguaje de especificación de valores – VSL, del perfil UML para sistemas embebidos de tiempo real – MARTE. Estas modificaciones permiten relacionar propiedades con operaciones de una clase, lo que facilita especificar propiedades no funcionales – NFPs como tiempo de respuesta, rendimiento, entre otras. Por último, se propone una nueva sintáxis para la representación de los NFRs en el diagrama de clases.

En esta tesis, se propone la especificación de NFRs temporales en el diagrama de clases. Para llegar a la especificación deseada, es necesario recorrer una etapa previa que consiste en la educción de requisitos. En este proceso de educción de requisitos, puede requerirse un método ágil o uno altamente documentado, como por ejemplo UN-METODO [Arango:2006], ¿cuál seleccionar?, dependerá de la aplicación y su complejidad.

5.1. Metodología para la educción de requistos

Debido al tipo de aplicaciones que se han explorado, no se ha requerido un método de educción de requisitos altamente documentado, ya que el proceso de identificación de las necesidades del cliente puede ser llevado a cabo utilizando el marco de trabajo basado en NFR [Chung:2000] o los casos de uso de UML [Booch:2006] [Hamilton:2006]. Sin embargo, es posible enmarcar la especificación de requisitos temporales no funcionales dentro de UN-METODO [Arango:2006], RUP [Jacobson:2000], CDM [Oracle:2000], XP [Beck:2000], FDD [Coad:1999], o cualquier otra metodología de desarrollo software, simplemente introduciendo pequeñas variaciones al método.
Tomando a UN-METODO, se construyen diagramas que están fundamentalmente enfocados en el problema y en la terminología del área, y no en la solución informática. El diagrama de actividades general del método propuesto puede verse en la Figura 5.1. Este diagrama se corresponde con el propuesto en UN-METODO [Arango:2006], con un par de modificaciones confinadas en los sub-

procesos del proceso objetivo del área, y en el proceso esquema conceptual. En esta tesis, se propone incorporar en el proceso objetivo del área el marco de trabajo basado en requisitos no funcionales [Chung:2000], que como se dijo en la sección 3.2.2, se enfoca en la identificación, descomposición y operacionalización de los NFRs, lo que a posteriori permite seleccionar y soportar las decisiones tomadas.

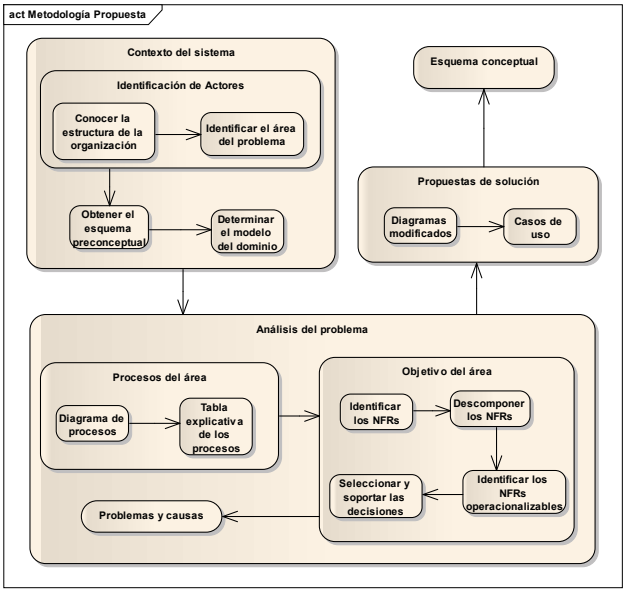

Figura 5.1: Actividades realizadas en el marco de UN-METODO con las modificaciones propuestas

Los resultados obtenidos en el proceso objetivo del área siguen siendo entradas válidas al proceso problemas y causas, ya que los softgoal operacionalizables del proceso objetivo del área, pasan a ser los objetivos de este, y así es posible determinar los problemas del área y que los causa. Debe tenerse en cuenta que los softgoal resultantes en esta etapa del desarrollo de software, pueden ser parcialmente satisfechos, y esto es lo que esencialmente los hace diferentes a los objetivos tradicionales.
El esquema conceptual especifica los objetos del sistema, los aspectos estructurales y dinámicos de los objetos del sistema que garantizan el comportamiento propuesto, y las restricciones sobre los objetos del sistema que son necesarias para garantizar el comportamiento propuesto.

5.1.1. Método para la representación y especificación de NFRs temporales

El núcleo fundamental del método propuesto en esta tesis para la especificación de NFRs en el diagrama de clases, se centra en la fase del esquema conceptual de UN-METODO (véase Figura 5.2). Todos los diagramas resultantes en esta fase utilizan el perfil UML MARTE [MARTE:2009] para detallar los modelos.

En este método inicialmente se definen los casos de uso, los cuales especifican lo que el sistema hace, y no cómo lo hace. Posteriormente se continua con el diagrama de actividades, el cual permite mostrar las acciones de alto nivel que encadenadas sirven para representar los procesos que ocurren dentro del sistema.

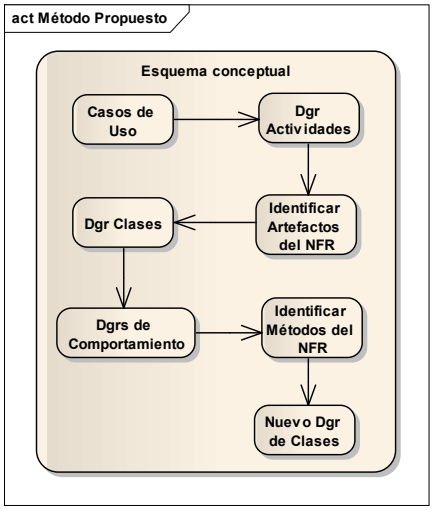

Figura 5.2: Actividades realizadas en el marco del método propuesto durante la fase del esquema conceptual

Con el diagrama de actividades y lo realizado con el marco de trabajo basado en NFRs, se identifican los artefactos UML que se puedan asociar con los NFRs del sistema, los cuales se descomponen en clases que permitirán crear la primera versión del diagrama de clases. El diagrama de clases así obtenido, se corresponde con el tradicional diagrama de clases de UML, en el cual no es posible especificar propiedades de los métodos.
El comportamiento dinámico del sistema debe incluir los diagramas que sean necesarios para la especificación clara y completa, esto es, el diagrama de secuencias, estados, tiempo, comunicación, entre otros. De estos diagramas, deben identificarse las operaciones que están asociadas con las clases que hacen parte de los artefactos relacionados con los NFRs del sistema. Por ejemplo, en el diagrama de secuencias los mensajes pueden invocar las operaciones de una clase, igualmente puede suceder en el diagrama de comunicación y el de tiempo. Sobre las operaciones identificadas como parte de los NFRs, se establecen las propiedades que posteriromente se representarán en el nuevo diagrama de clases.
Este nuevo diagrama de clases entregará al lector la información sobre los NFRs del sistema, utilizando la representación y especificación de NFPs propuesta.

5.2. Especificación formal de NFRs en el diagrama de clases

La representación de los NFRs en el diagrama de clases debe contener los aspectos de sintaxis y semántica. La sintaxis es la forma como se dibujará y escribirá la especificación, mientras que la

semántica habla del significado de dicha especificación.
La semántica estará soportada por la modificación del metamodelo del diagrama de clases de UML y el diagrama de expresiones de VSL definido en el perfil UML MARTE, mientras que la sintaxis se apoyará en los símbolos de uso corriente en UML y VSL [UMLSuper:2009] [MARTE:2009].

5.2.1. Propuesta semántica

La propuesta semántica debe ir en dos frentes: uno tiene que ver con la representación del NFR en el diagrama de clases, esto es, con la notación que utiliza UML; la otra tiene que ver con la especificación textual del requisito, que en esta tesis se hará según VSL. Con esto, lo que se pretende es que cualquier propiedad definida en MARTE [MARTE:2009], a la que se le permita mapearse a operaciones de una clase, pueda representarse en el diagrama de clases del sistema de una manera completa, formal y sin ambigüedades (para algunas NFPs véase la tabla 2.2).

5.2.1.1. Propuesta de modificación del metamodelo del diagrama de clases de UML

Como se presentó en la sección 2.4.2.1, las clases tienen asociadas propiedades y operaciones, y cada una de estas está relacionada con otros elementos del metamodelo del diagrama de clases (véase Figura 2.7). Por ejemplo, la metaclase `Operation`, que implementa una característica comportamental del sistema, la cual se infiere de la relación de herencia que tiene con la metaclase `BehavioralFeature`, tiene relaciones de composición con las metaclases `Parameter` y `Constraint`, lo que quiere decir que un parámetro y una restricción no pueden existir sin la operación. La asociación dirigida con la metaclase `Type`, permite asociar un tipo al dato retornado por la operación.
La metaclase `Property`, que es una característica estructural según la relación de herencia con la metaclase `StructuralFeature`, se relaciona con la metaclase `Association` y la metaclase `ValueSpecification`, de esta última se puede obtener el valor de la propiedad en el modelo.
De las relaciones establecidas en el metamodelo del diagrama de clases, no puede inferirse una relación entre las operaciones y las propiedades, esto es, las metaclases `Operation` y `Property`. Si se quieren representar los NFRs en el diagrama de clases, se hace necesario establecer una relación entre estas dos metaclases.

Esta tesis propone una modificación al metamodelo del diagrama de clases de UML, estableciendo una relación de tipo composición dirigida entre las metaclases `Property` y `Operation` (véase Figura 5.3). Con esta propuesta, las operaciones de las clases tendrán atributos propios, los cuales permitirán representar propiedades como el tiempo de ejecución de una operación.

La figura 5.3 muestra el metamodelo resumido del diagrama de clases. En este metamodelo se propone la nueva relación entre las metaclases `Property` y `Operation` (relación encerrada por

un óvalo). La relación propuesta es del tipo composición dirigida, donde la propiedad no puede existir sin la operación. Los extremos de esta nueva relación son:

- `operation`, representa la operación sobre la cual se especificará la propiedad.
- `ownedAttribute`, indica la propiedad de la operación que se especificará.

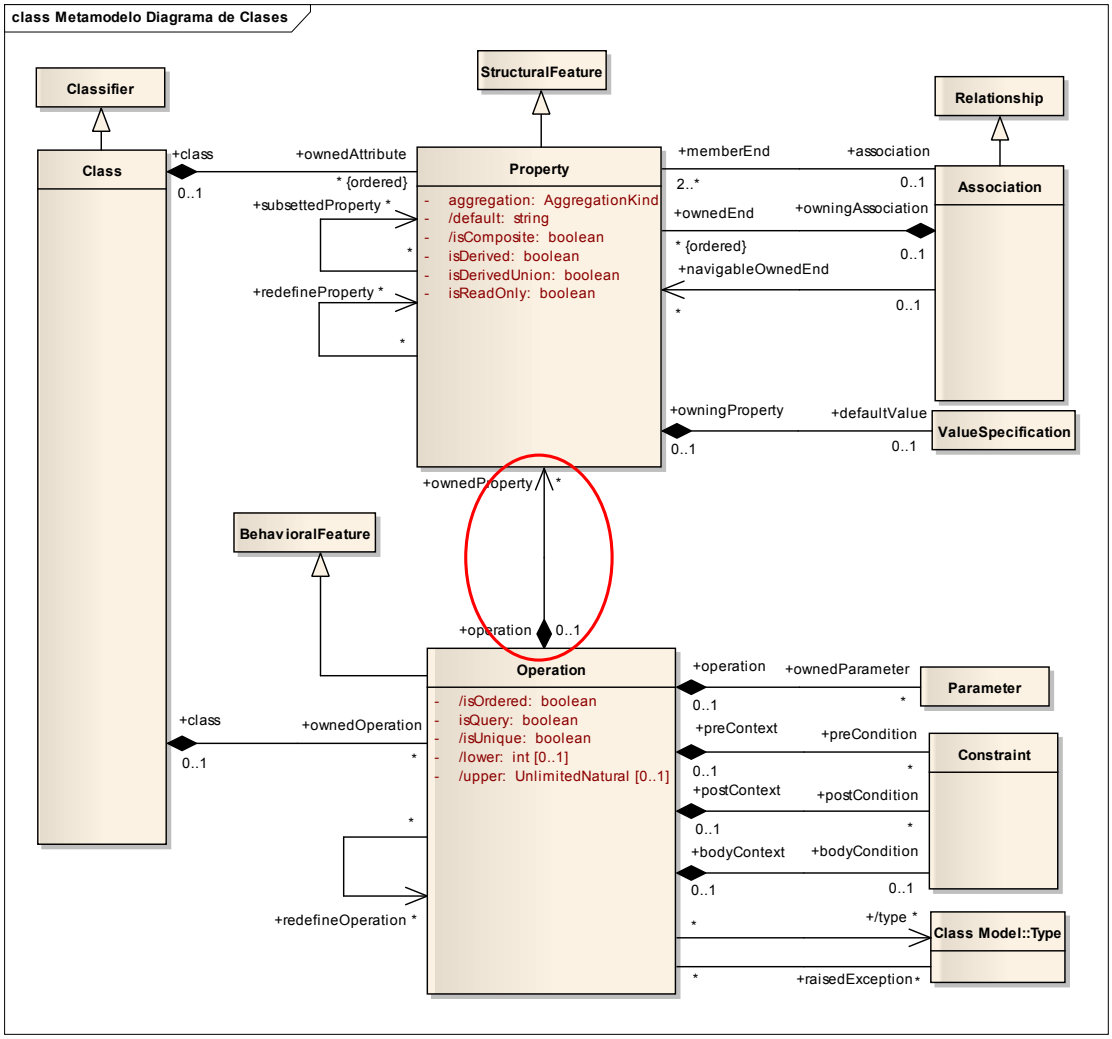

Figura 5.3: Propuesta de modificación del metamodelo del diagrama de clases (adaptado de [UMLSuper:2009])

5.2.1.2. Propuesta de modificación del metamodelo del diagrama de expresiones de VSL

Con el fin de completar la especificación de NFRs en el diagrama de clases, es necesario establecer una sentencia que formalmente defina las propiedades de los métodos de las clases. Para este

análisis, debe existir un contexto en el modelo UML. Este contexto debe identificar los elementos que son de interés para el análisis que se pretende realizar.
En MARTE, la definición del contexto se realiza con la metaclase `GaAnalysisContext` del perfil GQAM [MARTE:2009], la cual hereda de `ExpressionContext` del paquete `Expressions` de `VSL` (véase Figura 5.4).

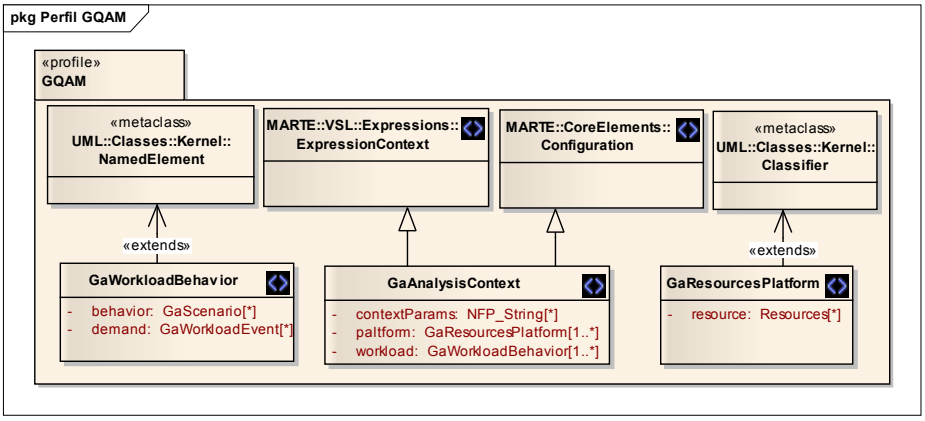

Figura 5.4: Extensiones UML para los estereotipos de nivel superior del perfil GQAM [MARTE:2009]

La definición del paquete `VSL::Expressions` permite establecer expresiones de contexto para las variables, pero no para las propiedades, y mucho menos para las propiedades de las operaciones, porque esta relación es propuesta en este tesis, y no existe aun en el metamodelo del diagrama de clases de UML [UMLSuper:2009]. Para poder expresar los NFRs sobre el diagrama de clases, es necesario establecer una relación entre las metaclases `Property` y `ExpressionContext`.

Esta tesis propone una modificación al metamodelo del diagrama `VSL::Expressions` del perfil UML MARTE [MARTE:2009], estableciendo una relación del tipo asociación dirigida entre las metaclases `Property` y `ExpressionContext` (véase Figura 5.5). Con esta propuesta, se podrán definir contextos para expresiones sobre las propiedades. Originalmente, las propiedades sólo están ligadas a las clases, pero por la modificación propuesta en esta tesis en la sección 5.2.1.1, relación de tipo composición entre las metaclases `Property` y `Operation` del metamodelo del diagrama de clases, se podrán definir igualmente contextos para propiedades de operaciones de una clase.

La asociación de tipo composición dirigida entre las metaclases `Property` y `Operation` del metamodelo del diagrama de clases propuesta en la sección 5.2.1.1, puede verse en la Figura 5.5 encerrada con un rectángulo, mientras que la nueva asociación entre las metaclases `Property` y `ExpressionContext` en el metamodelo del diagrama `VSL::Expressions` del perfil UML MARTE, puede verse en la Figura 5.5 encerrada con un óvalo. Los extremos de la asociación propuesta son:

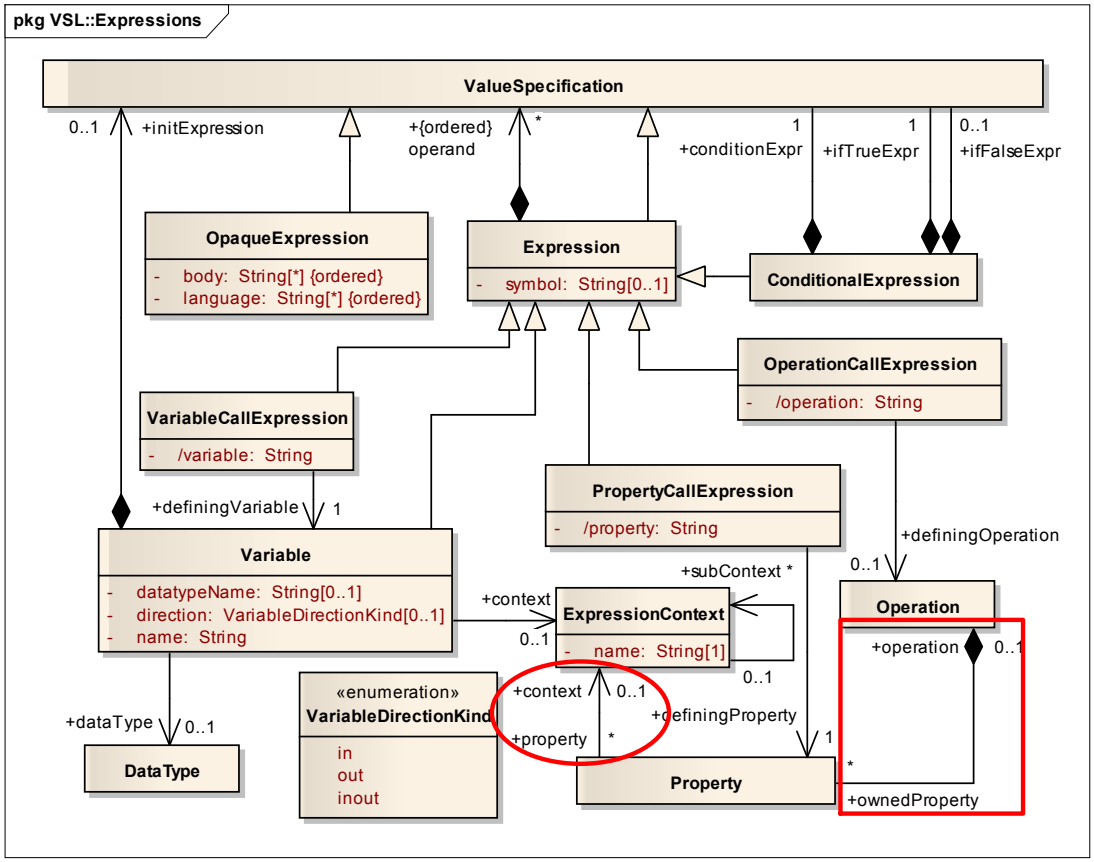

Figura 5.5: Propuesta de modificación del metamodelo para la construcción de expresiones con VSL (adaptado de [MARTE:2009])

- `context`, indica el contexto de interés sobre el cual se van a especificar las propiedades. Este contexto de interés son elementos del modelo UML, en este caso, como se trata de propiedades de las operaciones de una clase, el contexto debe apuntar a una operación definida en una clase del modelo.
- `property`, representa el atributo que se va a especificar de la operación de la clase definida en el contexto.

5.2.2. Propuesta sintáctica

La nueva sintáxis para representar clases, operaciones y atributos de las operaciones en el diagrama de clases, debe expresarle al lector en forma clara lo que se quiere decir, esto es, para el lector la simbología y las expresiones utilizadas no pueden dar lugar a confusiones o ambigüedades, de tal forma que el sistema quede claramente representado.
Para obtener una representación formal de NFRs en el diagrama de clases, es necesario representar

dichos NFRs utilizando los símbolos clásicos de UML, esto es, cajas con compartimientos, más una expresión textual que de forma precisa y utilizando un lenguaje formal especifique el NFRs.
Cuando se crea un diagrama UML se están instanciando las metaclases y las relaciones entre ellas que se definieron en el metamodelo. La forma clásica como el usuario visualiza estas instancias es la llamada representación gráfica (véase Figura 2.6), en la cual se representa la clase como un rectángulo con tres compartimientos, uno para el nombre, otro para los atributos y el restante para las operaciones. Debe tenerse en cuenta que esta no es la única representación gráfica posible para una clase, pueden existir tantas como se quiera [Muneton:2008], pero para esta tesis sólo esta representación es de interés. Sin embargo, también es posible instanciar la clase sin representación gráfica. Tomemos como ejemplo una versión simplificada de la clase `clsEndDevice` de la figura 2.6; la Figura 5.6 muestra la versión simplificada de la clase con representación gráfica, en la cual se identifican dos atributos y una operación, todo confinado dentro de una misma caja, lo que facilita el manejo del lenguaje. Igualmente, se muestra la misma clase sin representación gráfica, esto es, cada atributo y la operación se representan en cajas independientes, mostrando de manera explícita las relaciones de estas con la clase.

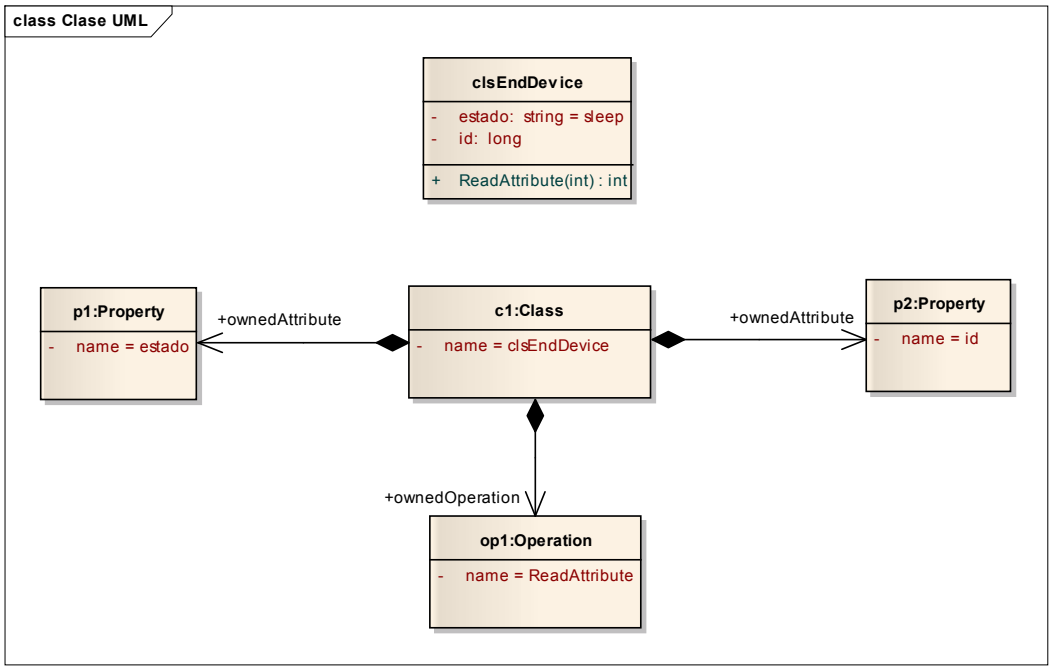

Figura 5.6: Clase UML con y sin representación gráfica

En la Figura 5.6 se instanció la clase `clsEndDevice` según el metamodelo del diagrama de clases de la Figura 2.7, esto es, el diagrama de clases sin la modificación del metamodelo propuesto en esta tesis. Por lo tanto, si las operaciones de la clase `clsEndDevice` tuvieran propiedades asociadas, este metamodelo no permitiría representar dichas propiedades.
Remitiéndonos a la Figura 5.3, si se quiere una representación gráfica de la metaclase `class` te-

niendo en cuenta la modificación propuesta, el rectángulo debe contener, además de los compartimientos para los atributos y las operaciones de la clase, un compartimiento para los atributos de las operaciones. Esto exigiría que por cada operación de la clase, debería existir un compartimiento que alojara sus atributos, lo que haría necesaria la identificación de cada compartimiento, y por ende bastante engorroso el manejo del símbolo. Por ejemplo, si la clase `clsEndDevice` tiene definidos dos atributos para la operación `ReadAttribute`, tales como tiempo de ejecución, `respTime`, y frecuencia de ejecución de la operación, `throughput`, estos atributos deberian estar dentro de un compartimiento del rectángulo de la clase que esté asociado a la operación `ReadAttribute`.
Sin embargo, la instanciación sin representación gráfica de la `clsEndDevice` con los dos atributos arriba mencionados para la operación `ReadAttribute`, y considerando la modificación al metamodelo del diagrama de clases propuesto en esta tesis, se puede ver en la Figura 5.7, nótese que la metaclase `Property` está asociada tanto a la clase como a las operaciones de esta.

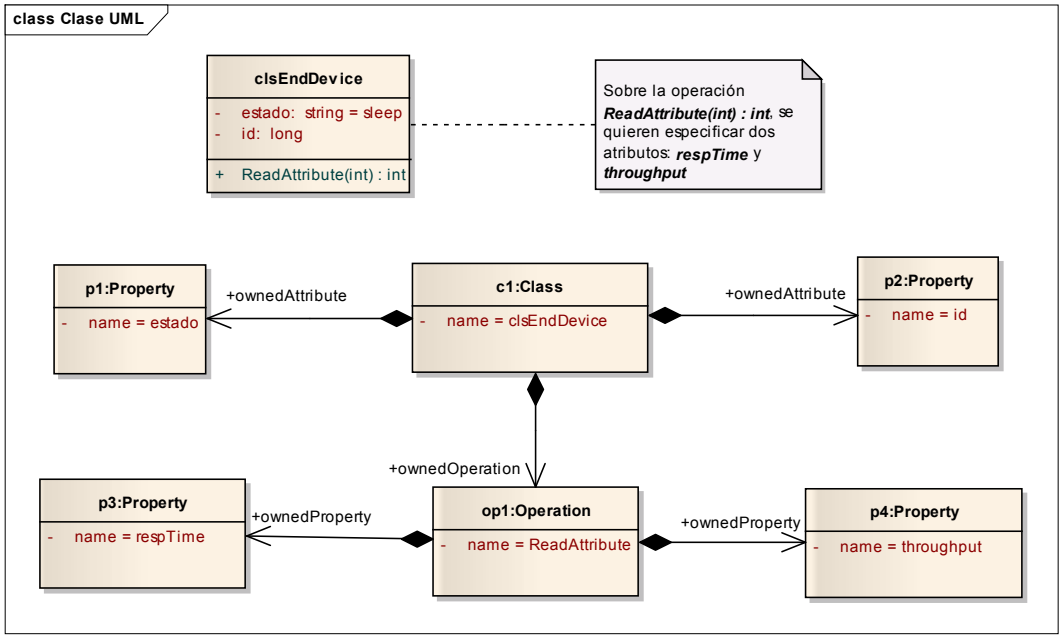

Figura 5.7: Clase UML con la modificación del metamodelo y sin representación gráfica

Con el fin de mantener una representación lo más compacta posible, pero que a su vez facilite la representación de los atributos de las operaciones, se propone una sintaxis combinada, esto es, que tenga una representación gráfica para la clase, sus atributos y las operaciones que no tienen propiedades asociadas, y una representación no gráfica para las operaciones que tienen propiedades asociadas. Por lo tanto, una representación más compacta de la Figura 5.7 puede verse en la Figura 5.8, nótese que los atributos de la clase y las operaciones de esta que no tienen propiedades asociadas[1], están en compartimientos del rectángulo que representa la clase, mientras que las operaciones de la clase que tienen propiedades asociadas tiene una representación no gráfica.

[1] En este caso no hay operaciones sin propiedades asociadas

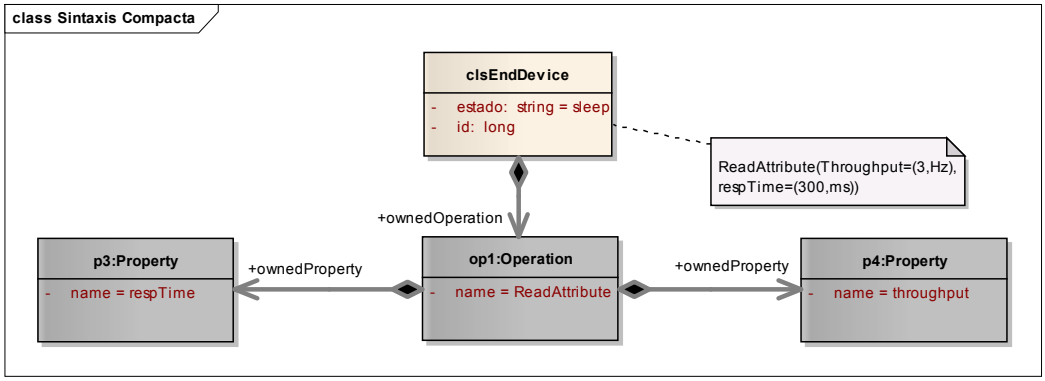

Figura 5.8: Clase UML con una sintaxis combinada e incluyendo la modificación del metamodelo

La sintaxis que se propone para la expresión textual que especifica formalmente las propiedades de una operación de una clase es:

```
<<GaAnalysisContext>>
Name_Op(Name_Prop1=(value1,unit), Name_Prop2=(value2,unit) ... ,
        Name_PropN=(valueN,unit))
```

En la sintaxis de la especificación textual, inicialmente se define el contexto de análisis utilizando la palabra reservada del perfil UML MARTE `GaAnalysisContext`; posteriormente se utiliza la sintaxis del tipo de dato `ChoiceType` de `VSL` [MARTE:2009], esto es, el nombre de la operación, con su espacio de nombres si fuera necesario, seguido del nombre de la propiedad, el signo '=' y el valor de esta, el cual será una tupla entre paréntesis que indique el valor separado por una coma de la unidad de medida. Cada propiedad debe estar separada de la siguiente por una coma, y el grupo de propiedades encerradas entre paréntesis.
La especificación textual debe ir dentro de una nota enlazada o no a algún elemento del diagrama de clases. Dependiendo del elemento al cual se asocie la nota, será necesario explicitar o no el espacio de nombres de la operación sobre la cual se especificarán las propiedades, esto con el fin de evitar conflictos entre operaciones.
Con lo anterior, la notación propuesta para representar las propiedades de las operaciones de una clase puede verse en la Figura 5.9.

Esta tesis propone para la representación de las propiedades de las operaciones de una clase, una sintaxis consistente de un componente gráfico y otro textual. El componente gráfico son varias cajas, una para la clase, una por cada operación sobre la cual se desee especificar una propiedad y una por cada propiedad de la operación.
La caja de la clase tiene un compartimiento para su nombre, otro para sus atributos y un último para las operaciones sobre las que no se requiera especificar propiedades. La caja que representa una operación sólo tendrá dos compartimientos, uno para el nombre de la operación con el tipo de dato por esta retornado, y el otro para los parámetros de la operación. La caja que

representa la propiedad de la operación que requiera especificarse, contendrá igualmente dos compartimientos, uno para el nombre de la propiedad, y el otro para el valor de la propiedad especificada.
La caja que representa la operación debe estar estereotipada con `Operation`, para indicar que se refiere a una operación de una clase, y `GaScenario`, para indicar que la operación es un escenario de comportamiento el cual tiene asociados recursos y propiedades. La caja que representa la propiedad debe estar estereotipada con `Property`, para indicar que se refiere a una propiedad de una operación. Ambas cajas, las que representan las operaciones y las que representan las propiedades, se dibujan con un sombreado gris, esto con el fin de ayudarle al lector a identificar que dichas cajas no representan clases, sino operaciones de clases y propiedades de operaciones de clases.
La relación entre la clase y la operación, y la operación y la propiedad, se dibuja como relación de composición dirigida en un tono gris y más gruesa que la relación de composición clásica, con el fin de indicarle al lector que esta no es una relación entre dos clases, sino entre la clase y su operación, y entre la operación y su propiedad.

Con esta nueva sintáxis, se está creando un nuevo diagrama de clases que permite combinar operaciones con algunas propiedades que hablan de esta. Esta representación le da mayor expresividad al modelo, permitiendo conocer elementos que, antes de esta forma de representación, no se podian inferir de uno de los diagramas estructurales más importantes de un modelo UML, el diagrama de clases.
Al estar las operaciones estereotipadas con `GaScenario`, cualquier propiedad definida en MARTE [MARTE:2009] que haga parte de los atributos cuantitativos de este estereotipo, puede, con esta nueva propuesta sintáctica, representarse formalmente como una nota en el diagrama de clases del sistema.

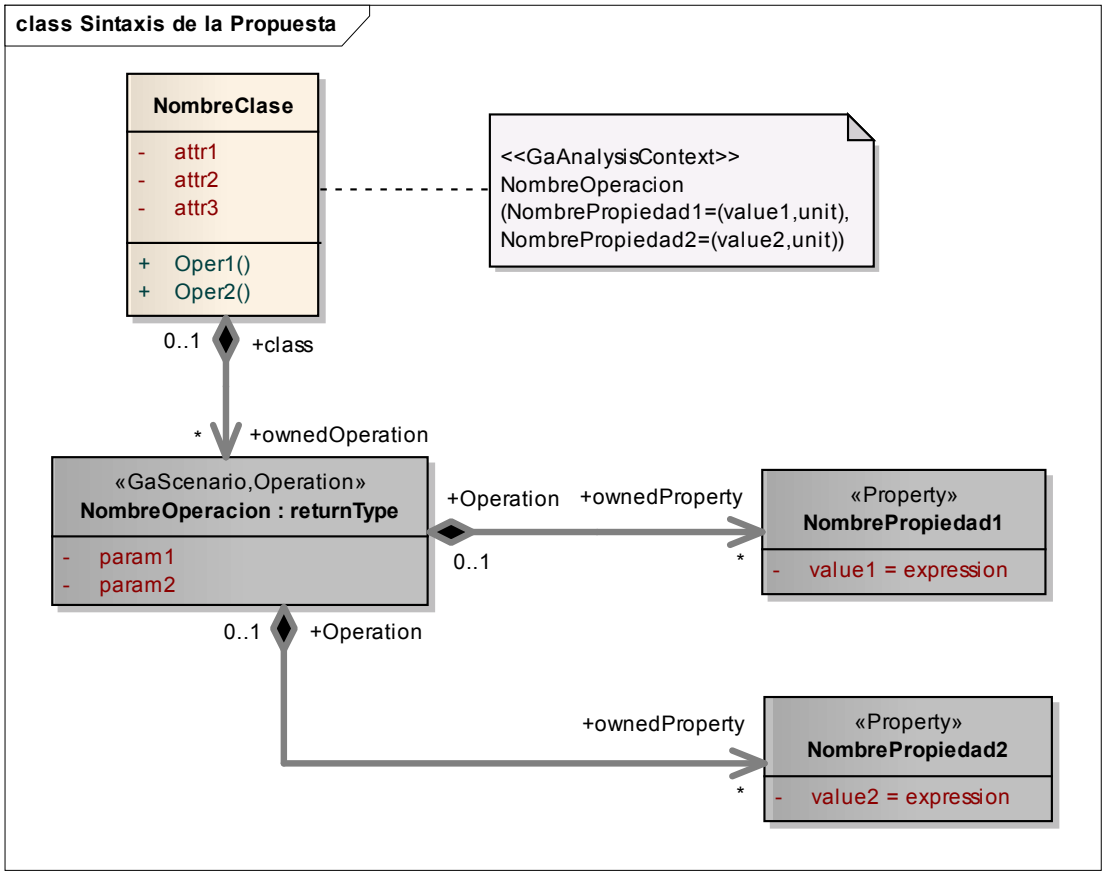

Figura 5.9: Sintáxis para la representación de propiedades de las operaciones de una clase en el diagrama de clases de UML

Capítulo 6
Resultados y discusión

Resumen En este capítulo se proponen dos casos de estudio de sistemas embebidos de tiempo real. Primero, se visualiza la especificación del sistema en el diagrama de clases de UML tradicional, y posteriormente se muestra la especificación del sistema con el nuevo diagrama de clases propuesto en esta tesis. De esta forma, se destacan las bondades del método propuesto.

En este capítulo, se presenta la aplicación del método propuesto en esta tesis para especificar NFRs en el diagrama de clases de UML. Se toman dos casos de estudio de sistemas embebidos con uno o varios NFRs. Los casos de estudio son: el sistema de vigilancia para un campamento militar y el sistema de monitoreo del factor de potencia de un motor DC utilizando un gateway Ethernet - ZigBee.

6.1. Caso de estudio: sistema de vigilancia para un campamento militar

Este caso de estudio se presentó como ejemplo en la sección 3.1, y se especifico de diferentes formas en todo el capítulo 3. Por facilidad reproduciremos su enunciado.
Consideremos un sistema de seguridad para un campamento militar. Este sistema de seguridad tiene como núcleo principal una red inalámbrica de sensores, en adelante denominada red inalámbrica de área personal (Wireless Personal Area Network – WPAN). Los nodos de la WPAN son de dos clases, coordinadores y dispositivos finales, cada WPAN puede sólo tener un coordinador, pero múltiples dispositivos finales. El coordinador se encarga de iniciar la WPAN, asignar las direcciones a cada dispositivo final, servir como puente de comunicación entre nodos finales si fuera necesario, aceptar, rechazar o cancelar la vinculación de un dispositivo final a la WPAN por él administrada, entre otras funciones. Los dispositivos finales tienen asociados detectores de movimiento o sensores de calor, los cuales permiten determinar la presencia de intrusos. Esta información debe ser enviada al coordinador con el fin de que éste la centralice y tome la decisión de activar o no la alarma general de seguridad del campamento, la cual está asociada a otro dispositivo final.
El sistema de seguridad es un sistema de tiempo real, ya que tiene varias exigencias que se enuncian a continuación:

- Se considera que un dispositivo final está activo cuando cualquiera de los sensores a él adjunto se activa
- El coordinador consultará cíclicamente a los dispositivos finales acerca de su estado. El tiempo que transcurre entre una consulta y otra no debe ser mayor a 1*s*
- Los dispositivos finales consultarán sus sensores sólo a petición del Coordinador
- Luego de detectada la presencia de intrusos por parte del dispositivo final, el sistema debe garantizar que la alarma se active en menos de 500*ms*

La topología de la WPAN puede verse en la Figura 3.1. Aquí se distribuye la red entre un nodo coordinador, y cuatro dispositivos finales; tres utilizados para sensar y un cuarto para activar la alarma. El sistema sólo posee detectores de movimiento y sensores térmicos, ambos infrarrojos. Un dispositivo final cuenta con un sensor y un detector, mientras que los otros sólo cuentan o con un sensor o con un detector.
En la sección 3.2.3.3 se especificó el sistema de la WPAN utilizando UML y MARTE, allí se obtuvieron los diagramas de comportamiento del sistema. El diagrama estructural más importante, el diagrama de clases, se obtuvo con la solución presentada en la 3.2.3.1, y corresponde a la Figura 3.13, la cual, por facilidad, se reproduce en esta sección en la Figura 6.1.

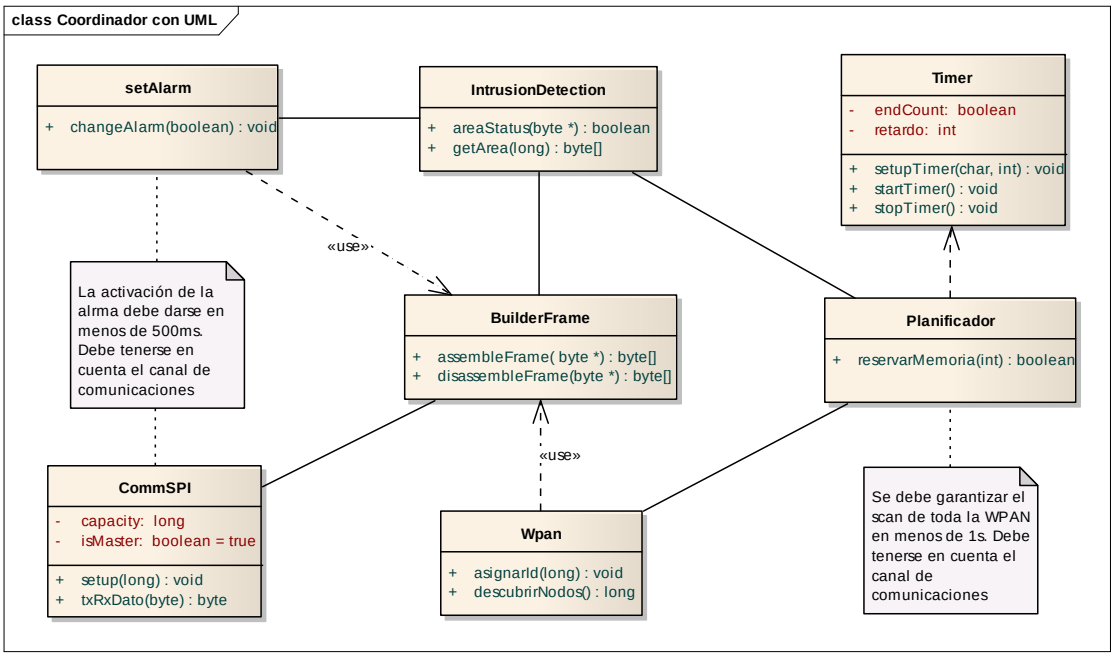

Figura 6.1: Diagrama de clases UML de la WPAN sin la especificación de los NFRs

Como se dijo arriba, en la sección 3.2.3.3 se especificó el sistema utilizando el perfil UML MARTE. En el diagrama de secuencia del Coordinador de la WPAN (véase Figura 3.21), se puede apreciar que las operaciones `assembleFrame` y `disassembleFrame` de la clase `BuildeFrame` están estereotipadas como `GaStep` [MARTE:2009], lo que permite asociarles algunas NFPs. En

ambas operaciones se especificó su tiempo de respuesta, `respT`, para `assembleFrame` es de 25*ms*, mientras que para `disassembleFrame` es de 30*ms*.

6.1.1. Implementación obtenida con el método propuesto en esta tesis

La especificación del NFP `respT` para las operaciones de la clase `BuildeFrame`, utilizando el método propuesto en esta tesis, puede verse en la Figura 6.2. Allí puede observarse que las operaciones `assembleFrame` y `disassembleFrame`, se separaron de la clase `BuilderFrame`. Cada operación tiene una caja estereotipada con `GaScenario` y `Operation`. El estereotipo `Operation` y el rol del extremo de la asociación indica que es una operación, y no una clase, así utilice un símbolo igual al de las clases. El estereotipo `GaScenario`, extractado del perfil UML MARTE [MARTE:2009], permite asociar a dicha operación NFPs que sirven para satisfacer los NFRs del sistema. La caja en la que se representa la operación tiene el nombre de la operación con su tipo de retorno, y adicionalmente tiene un compartimiento para los parámetros de la operación. En la relación de composición dirigida entre la clase y la operación se identifica claramente los roles y la multiplicidad.
Para la NFP `respT` se utiliza igualmente una caja con un compartimiento. El nombre de la caja es el de la NFP, mientras que en el compartimiento se ubican los atributos del NFP. Un NFP se puede representar con sólo el atributo `value` o con los atributos `value` y `unit`. Si se utiliza sólo el atributo `value`, el valor debe ser presentado como una tupla `(value,unit` para que no exista ambigüedad. Para la operación `assembleFrame` se presentó el valor de `respT` como un valor flotante de la forma `value: float = (25,ms)`. Mientras que si la representación se hace con dos atributos, `value` y `unit`, al primero se asocia el valor, mientras que al segundo la unidad del valor. Para la operación `disassembleFrame`, la NFP se especifica como `value: float = (30,ms)` y `unit: String = ms`.

La especificación textual representada en el diagrama de clases de la Figura 6.2 es una nota que, por facilidad en la lectura del diagrama, no está enlazada a ningún elemento del modelo. Sin embargo, como se refiere a las operaciones de la clase `BuilderFrame`, en la expresión del contexto, `GaAnalysisContext`, se indica el espacio de nombres completo para la operación, esto es, con el nombre de la clase incluído.

6.2. Caso de estudio: sistema de monitoreo Web del FP de un motor DC

En [Serna:2010a] se describe un sistema para monitoreo Web del factor de potencia – FP de un motor DC, el cual hace parte de un sistema de gestión de demanda de energia (DSM – Demand Side Management). El sistema está constituído por un gateway Ethernet - ZigBee y una WPAN. A través de un navegador, el usuario hace solicitudes de medición del FP, lo que el gateway traduce en mensajes ZigBee, cuya respuesta se envía de nuevo al usuario. El gateway Ethernet - ZigBee está constituido por un servidor Web embebido – EWS y la pila de protocolos ZigBee. Utilizando la pila de protocolos ZigBee, el EWS reporta el factor de potencia al browser que lo solicite. Este sistema,

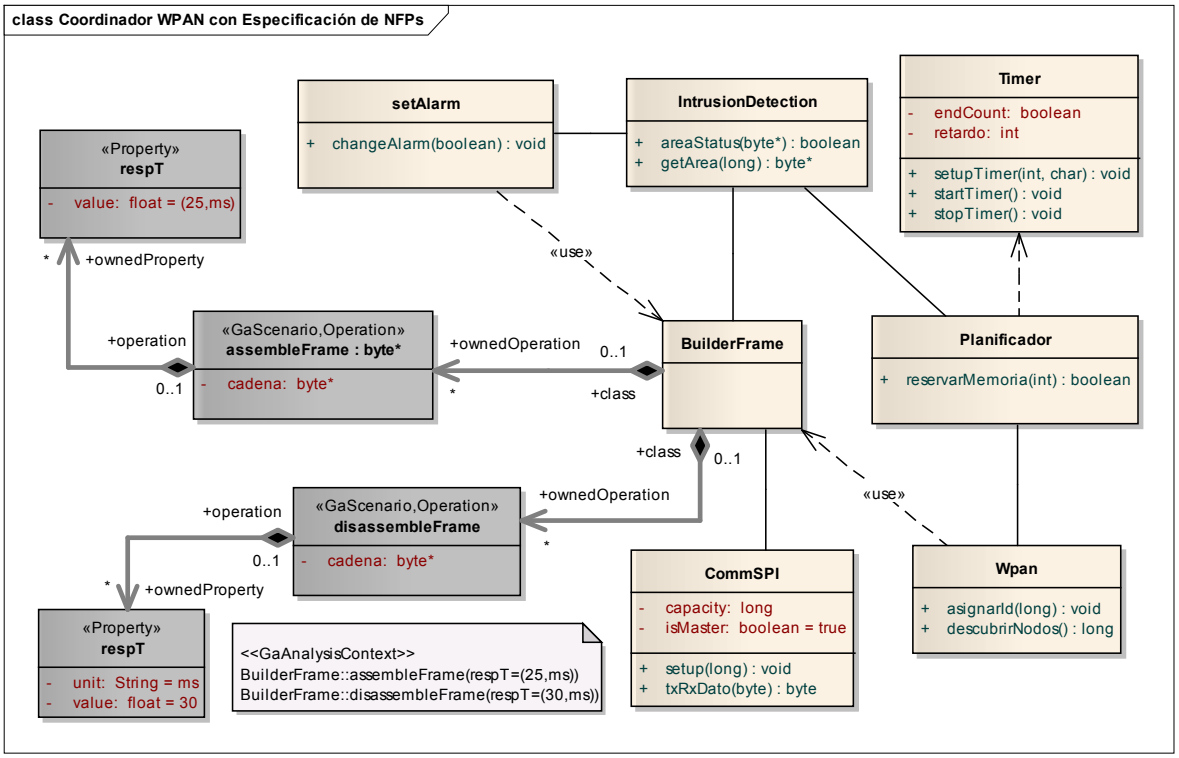

Figura 6.2: Diagrama de clases UML de la WPAN con la especificación de los NFRs

como se plantea en [Serna:2010a], tiene un requisito de tiempo real, y es que cada 5s el nodo final ZigBee debe estar reportando el valor del factor de potencia del motor DC al gateway, esto es, la frecuencia de envío debe ser de $0{,}2Hz$. Sólo con fines ilustrativos, se cambiará este requisito por uno más exigente, aunque sea menos realista para el sistema que se está considerando. El nuevo requisito es que el nodo final ZigBee reporte el factor de potencia, al gateway Ethernet - ZigBee, con una frecuencia de $10Hz$.

El diagrama de despliegue del sistema puede observarse en la Figura 6.3. Allí los nodos se estereotipan como dispositivos procesadores `GaExecHost` o enlaces físicos de comunicación `GaCommHost`. Sobres estos dispositivos se especifican ciertas NPFs como la capacidad y el tiempo de bloqueo del canal de comunicaciones, `capacity` y `blockT`, y la demanda de host para transmisión y recepción en el dispositivo final, `commTxOvh` y `commRcvOvh`.

El diagrama estructural más importante del gateway, el diagrama de clases, puede verse en la Figura 6.4. Allí se identifican librerías preconstruídas que facilitan las labores de comunicación del gateway, esto es, la pila ZigBee, la pila TCP/IP y el módulo SCI. Se identifican tres clases en dicho diagrama, una para la comunicación Ethernet, otra que hace las funciones de gateway, y la última para recibir la medición del factor de potencia, la cual posee la operación `getFP()`.

Para satisfacer el requisito temporal que se pide, debe tenerse en cuenta que es en el intercambio de información entre el gateway y el nodo final ZigBee donde deben especificarse los tiempos, esto

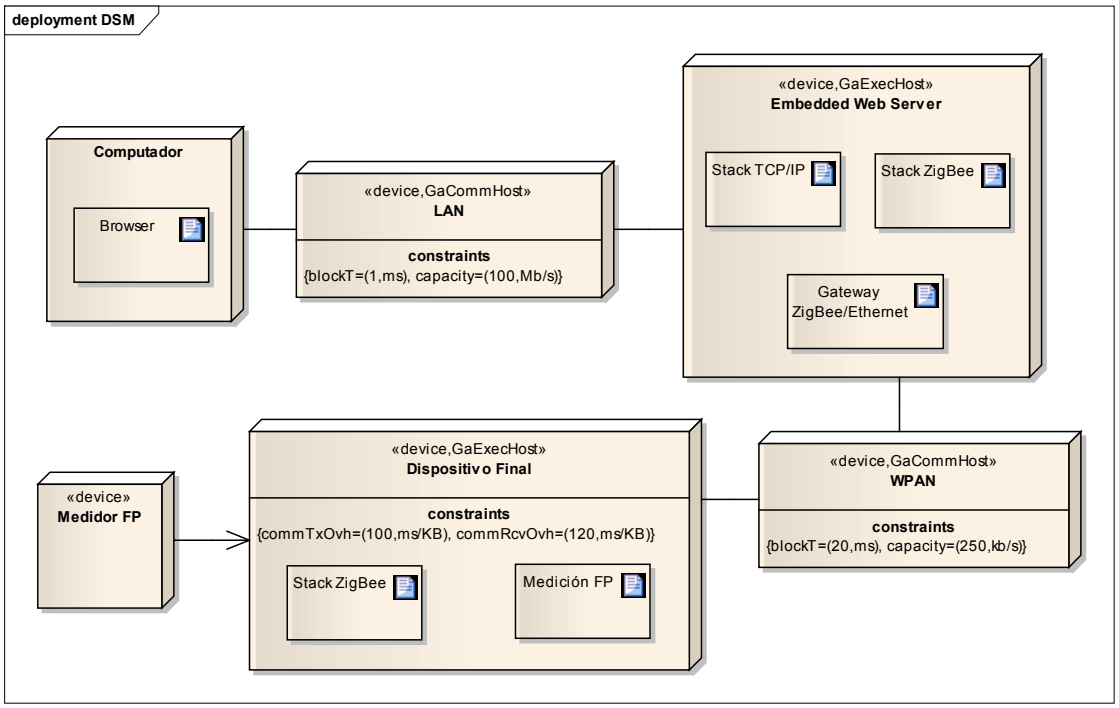

Figura 6.3: Diagrama de despliegue del gateway Ethernet-ZigBee [Serna:2010a]

es, en la WPAN. Con el fin de simplificar los cálculos, se considerará el tamaño de la trama como el máximo para la tecnología ZigBee, $128B$; e igualmente se considerará que la demanda de host para la transmisión y recepción en el nodo final y en el gateway es la misma. De acuerdo a esto, y con los parámetros dados en el diagrama de despliegue, podemos establecer el tiempo que invierte el sistema desde el instante que está disponible el valor del FP en el nodo final, hasta que dicho valor está disponible en el gateway, para ser procesado por este. Por lo tanto:

$$FTZ = 128 * 8 \frac{1}{0{,}25 * 250000} = 16{,}4ms \tag{6.1}$$

$$TT = 128 * \frac{0{,}1}{1024} = 12{,}5ms \tag{6.2}$$

$$RT = 128 * \frac{0{,}12}{1024} = 15ms \tag{6.3}$$

$$TotalTime = 43{,}9ms \tag{6.4}$$

FTZ es el tiempo que tarda el mensaje viajando por la red ZigBee (véase ecuación 6.1). En este caso, se considera que la velocidad de transmisión por la red es el 25% de la capacidad total de

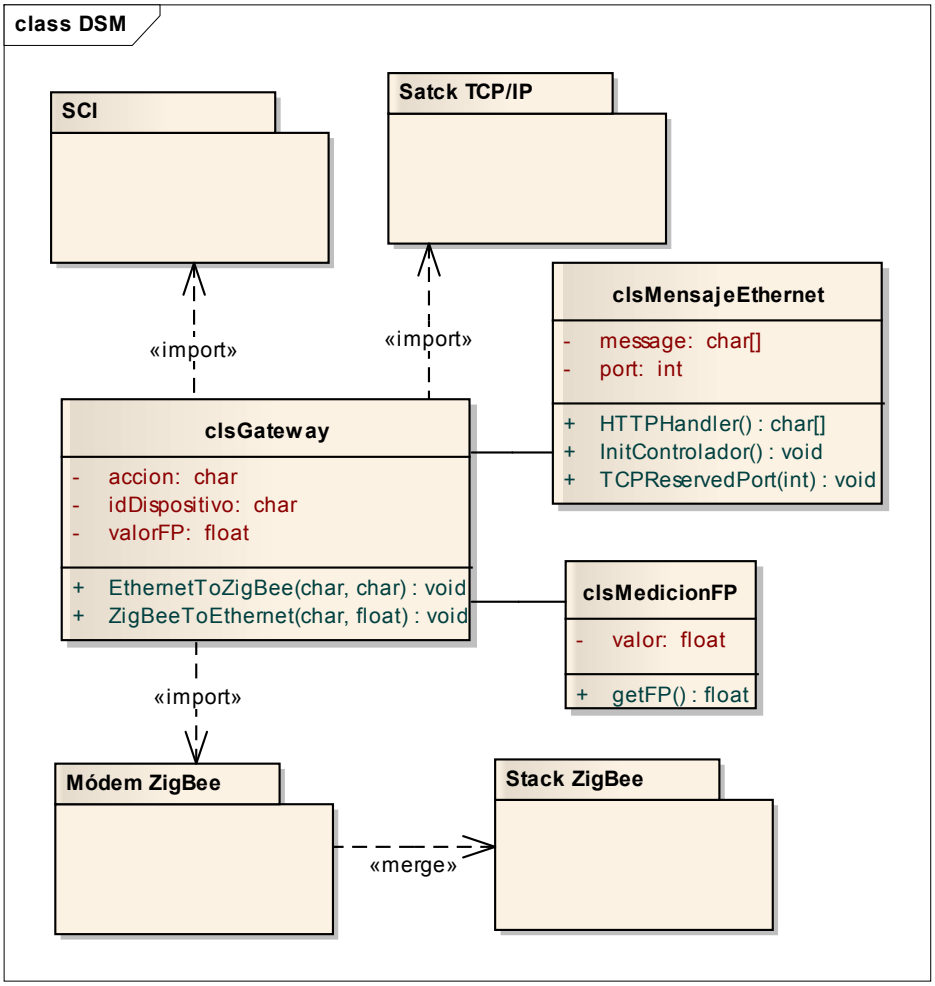

Figura 6.4: Diagrama de clases del gateway Ethernet-ZigBee sin la especificación del NFR [Serna:2010a]

la misma. Los tiempos de host para procesar los mensajes transmitidos y recibidos son `TT` (véase ecuación 6.2) y `RT` (véase ecuación 6.3). El tiempo total en el cual el mensaje se está transmisitendo o recibiendo está dado por `TotalTime` (véase ecuación 6.4). Ya que el requisito de tiempo para esta lectura cíclica es de `100ms`, se tienen `56.1ms` para repartir entre los tiempos de ejecución de las operaciones involucradas en el proceso, `SendFP()` en el nodo final y `getFP()` en el gateway. Debido a que el procesador del gateway, Rabitt 3000 [Rabbit:2010], tiene mayor poder de cómputo que el del nodo final, Freescale HCS08 [Freescale:2010], se darán 40*ms* para la ejecución de la operación en el nodo final, mientras que se darán 16,1*ms* para la ejecución de la operación en el gateway.

Con lo anterior, se pueden definir dos NFPs para la operación `getFP()`: una tiene que ver con el tiempo de ejecución de la operación, `respT`, establecido en 16, 1*ms*; mientras que la otra propiedad establece la frecuencia de ejecución de la operación, `throughput`, en 10*Hz*. Es claro que los

participantes en el diagrama de secuencia de la Figura 6.5, no tienen que ser necesariamente objetos software, como tradicionalmente se ha utilizado en UML 1.x; en este caso son partes del sistema, un concepto más orientado a un lenguaje de modelado de sistemas como lo es UML 2.x. Es por esta razón que no se identifican directamente las clases del gateway en el diagrama de secuencia, más bien se identifican el gateway mismo y el nodo final. Sin embargo, a pesar de no estar explíctas las clases de los nodos, en el llamado iterativo a la medición del FP y el respectivo envío, también iterativo, del FP al gateway, los nombres de los mensajes si involucran las operaciones de interés para nuestra NFP.

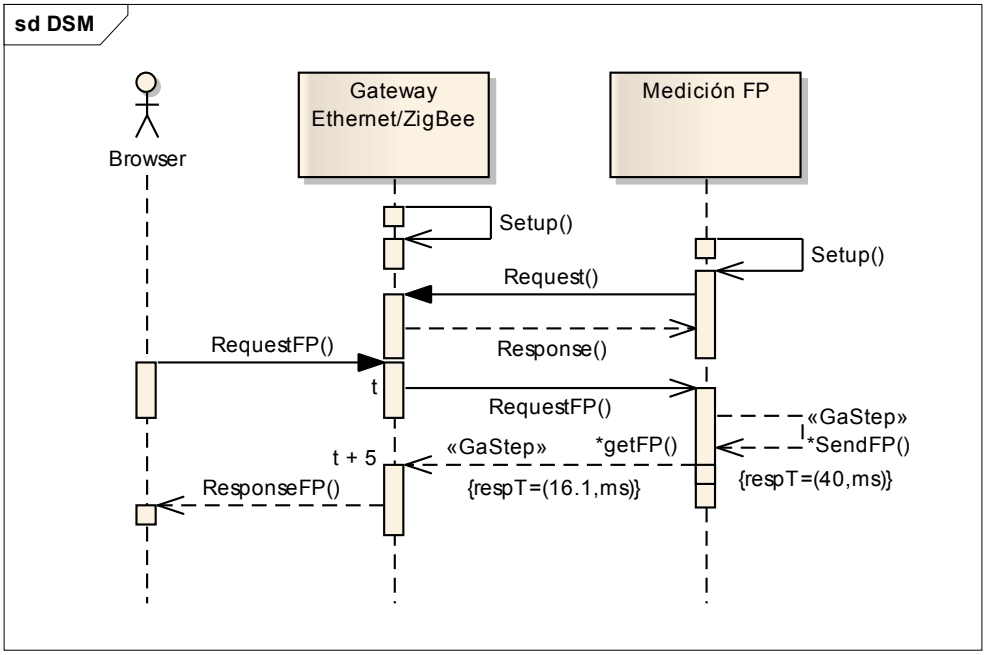

Figura 6.5: Diagrama de secuencia del gateway Ethernet-ZigBee [Serna:2010a]

Son dos las operaciones, de dos clases diferentes, a las cuales debe especificárseles la NFP: `getFP()` de la clase `clsMedicionFP` del nodo gateway, y `SendFP()` de una clase del nodo final ZigBee. Para poder asociar NFPs a las operaciones, es necesario, según MARTE, estereotiparlas como `<<GaStep>>`. De esta manera, se puede dar un valor a la NFP `respT` utilizando la sintaxis del VSL [MARTE:2009].

El diagrama de clases para el gateway del sistema de monitoreo del FP con la modificación propuesta en esta tesis, puede verse en la Figura 6.6. Allí se destacan, tanto en forma gráfica como textual, dos NFPs que contribuyen a satisfacer el NFR planteado en el sistemas.

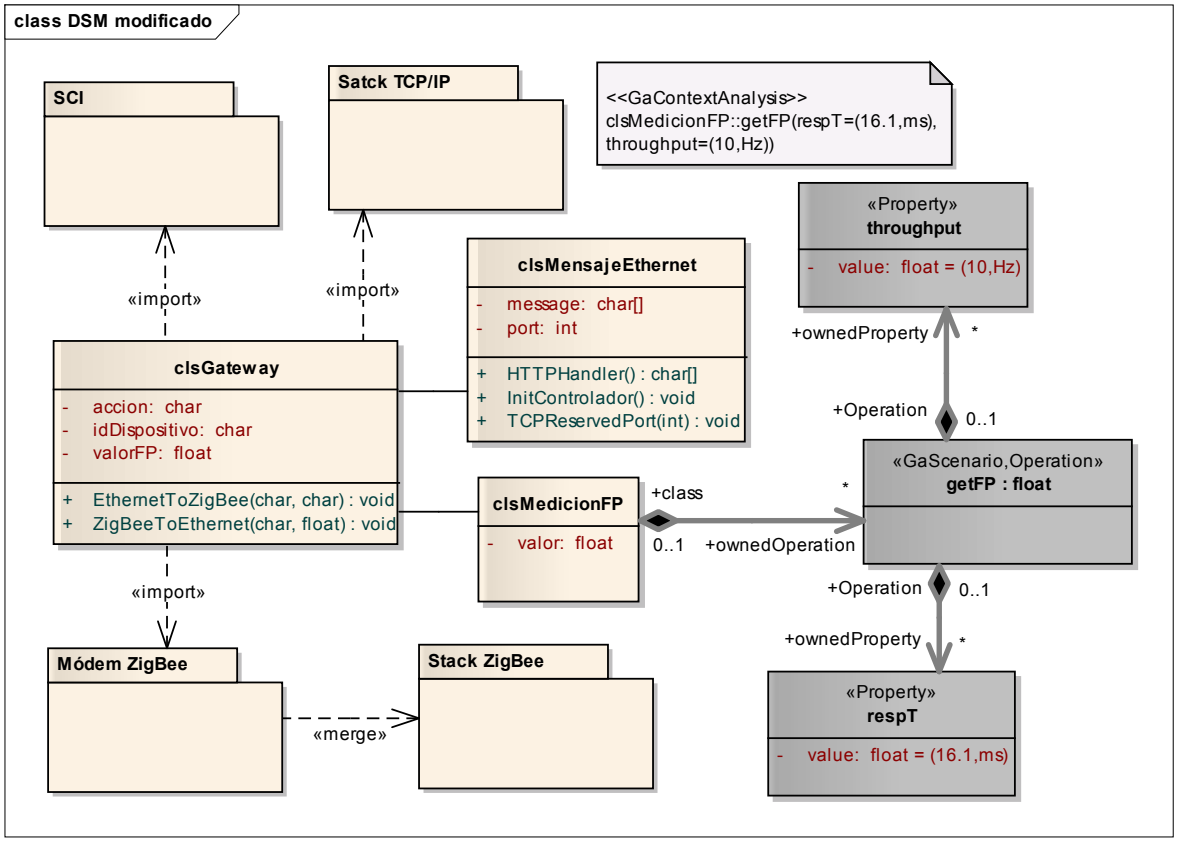

Figura 6.6: Diagrama de clases del gateway Ethernet-ZigBee con la especificación del NFR

6.3. Análisis de resultados

Luego de aplicar la propuesta presentada en esta tesis para la especificación de NFPs en el diagrama de clases, con el fin de satisfacer los NFRs del sistema, se pueden concluir varias cosas:

- El diagrama de clases resultante, luego de aplicar el método propuesto en esta tesis, tiene un nivel de detalle mayor que el diagrama de clases tradicional encontrado en los modelos UML. Esto se debe fundamentalmente a dos factores: uno, la representación gráfica y no gráfica de las clases; dos, a la nota adjunta al diagrama de clases, que utilizando VSL, especifica de manera formal propiedades sobre las operaciones de una clase.
- Las relaciones entre la clase y la operación, y la operación y sus propiedades, deben ser dirigidas, con el fin de identificar quién incluye a quién y además con el rol en cada extremo. Adicionalmente, la relación se dibuja en un tono gris y más gruesa que la relación de compo-

sición tradicional. Este nivel de detalle es necesario con el fin de no confundir las relaciones entre operaciones y clases, y entre propiedades y operaciones, como relaciones entre clases.

- Las cajas que sirven de símbolos para las operaciones y las propiedades deben dibujarse con un sombreado gris. Este detalle es necesario con el fin de no confundir las operaciones y las propiedades con clases.
- La especificación textual de las propiedades de las operaciones de una clase, utiliza un lenguaje con una sintaxis y una semántica claramente definida, lo que permite asociar sentencias al modelo con un significado libre de ambigüedades, y con una escritura de fácil lectura por modeladores o máquinas para procesamiento automático de sentencias.
- El diagrama de clases resultante, luego de aplicar el método propuesto en esta tesis, tiene representación gráfica y no gráfica de las clases, lo que puede generar confusiones si no se estreotipan las cajas, no se identifican los roles de los extremos de las asociaciones y no se modifica el dibujo de los símbolos. Por esta razón, cuando se construya el nuevo diagrama de clases, es mandatorio estereotipar adecuadamente las cajas que representan las operaciones y las propiedades de las operaciones, dibujarlas con tono gris e indicar el rol en cada extremo de las relaciones.
- El nivel de detalle del diagrama de clases resultante, permite tomar decisiones de implementación del sistema, tanto en aspectos del software, como el lenguaje de implementación o el algortimo a utilizar, como en aspectos del hardware, plataforma de ejecución, reloj de procesamiento, entre otros.
- Las NFPs que tienen que ver con el tiempo, restringen los aspectos hardware y software del sistema, ya que regularmente se requiere optimizar tanto el algoritmo como la plataforma de ejecución. Esto en ocasiones puede presentar incompatibilidades entre arquitecturas hardware y arquitecturas software, ya que, por ejemplo, no todas las arquitecturas hardware soportan lenguajes orientados a objetos.
- Como el modelado del sistema RT/E se realiza de acuerdo al paradigma de orientación a objetos, es necesario garantizar que el lenguaje de implementación sea orientado a objetos. En la mayoría de las plataformas de bajo perfil, el lenguaje nativo de trabajo es el ensamblador. Como lenguaje de alto nivel o nivel intermedio se utiliza C, el cual no es orientado a objetos. Sin embargo, dependiendo del ambiente de desarrollo integrado – IDE con el que se esté trabajando, será posible programar el sistema en C++ o Java.
- De esta forma, para tomar las decisiones de implementación del sistema, es necesario tener en cuenta varios elementos que son claves para el éxito del desarrollo: las NFPs especificadas en el diagrama de clases; la arquitectura hardware del sistema sobre el cual se va a desplegar la aplicación; el IDE con el cual se va a programar el procesador o los procesadores del sistema RT/E; el lenguaje de programación que se utilice para la implementación del código; el uso de sistemas operativos embebidos que puedan facilitar tareas pero igualmente ralentizar los procesos, esto debido a la sobrecarga de las llamadas a funciones y rutinas.
- Para garantizar un desempeño óptimo del sistema, es necesario revisar aspectos detallados del hardware, ya que es este el que permite que los tiempos de ejecución, el rendimiento, y demás NFPs puedan satisfacerse. Por ejemplo, detalles como tamaño de los registros de la

CPU, reloj del sistema, tiempo de acceso a memoria, conjunto de instrucciones, implementación de las instrucciones a nivel de microarquitectura, son aspectos claves que permiten determinar el desempeño de la plataforma hardware.

Capítulo 7
Conclusiones y trabajos futuros

Resumen En este capítulo se presentan los aportes más importantes de este trabajo. Se destaca la nueva propuesta semántica y sintáctica que permite especificar, en el diagrama de clases de UML, propiedas no funcionales presentes en requisitos no funcionales. Igualmente, dentro de un método de desarrollo de software, es posible lograr cierto grado de consistencia y trazabilidad en esta clase de requisitos

7.1. Conclusiones

Los aportes de este trabajo son los siguientes:

- Los requisitos temporales no funcionales, se han representado por medio de perfiles UML o extensiones propietarias en diagramas comportamentales del modelo. Este trabajo, permite representar los requisitos no funcionales en el diagrama estructural más importante, el diagrama de clases. De esta manera, es posible tomar decisiones de implementación al momento de llevar a cabo el desarrollo, ya que las propiedades no funcionales especificadas tienen que ver con la plataforma de ejecución, y diversos tiempos asociados a las operaciones. Todo esto impone restricciones acerca del lenguaje de construcción, la lógica de implementación o el procesador.
- Esta propuesta puede hacer parte de métodos de desarrollo de software tradicionales, integrándose con ellos y extendiéndolos fácilmente. Es posible inicialmente elicitar requisitos con el marco de trabajo basado en requisitos no funcionales. Posteriormente se pueden obtener los diagramas UML en los cuales se especifiquen la propiedades no funcionales utilizando el perfil MARTE. Finalmente representar estas propiedades no funcionales en el nuevo diagrama de clases.
- Para esta nueva representación, sólo es necesario un cambio en el metamodelo del diagrama de clases y otro cambio en el metamodelo del diagrama de expresiones de VSL del perfil UML MARTE. En el metamodelo del diagrama de clases se establece una relación de com-

posición, hasta ahora inexistente, entre las metaclases `property` y `operation`; en el metamodelo del diagrama de expresiones de VSL se establece una asociación dirigida entre las metaclases `property` y `ExpressionContext`.

- Para la notación de las propiedades de una operación, se utiliza la representación no gráfica de la clase y su operación. La relación entre la operación y sus propiedades se establece por medio de una relación de composición dirigida. Igualmente se utiliza una nota adjunta o no a la clase, donde se indique, utilizando VSL, la propiedad de la operación.
- No se introducen nuevos elementos al metamodelo, es suficiente con los que se tiene en el diagrama de clases y en el diagrama de expresiones de VSL, solo se requiere establecer una nueva relación entre dos de sus elementos.
- Por la combinación de elementos del modelo y el metamodelo en el nuevo diagrama de clases, se pueden expresar metadatos referentes a los métodos de una clase. Con esto se logra darle mayor expresividad al modelo, ya que se identifican elementos que no es posible inferir del diagrama de clases tradicional
- Las especificaciones cuantitativas pero informales de ciertos tipos de requisitos temporales no funcionales dadas por el cliente, pueden desagregarse en escenarios y operaciones que posteriormente se utilicen para garantizarán la consistencia del requisito, desde su representación en un grafo de interdependencia softgoal, pasando por los diagramas de actividades y secuencias, hasta llegar a las restricciones de implementación que el nuevo diagrama de clases le presenta al desarrollador.
- Entre el diagrama de clases definitivo del sistema, y el diagrama de actividades o el grafo de interdependencia softgoal, es posible verificar la trazabilidad del requisito temporal no funcional. Todo esto se debe al hecho de que las propiedades de las operaciones son explícitas en el diagrama de clases, y se corresponden con los requisitos temporales no funcionales especificados en la etapa de educción de requisitos.
- La representación en el diagrama de clases de las propiedades no funcionales, utiliza UML estándar y el perfil MARTE del OMG (Object Management Group). Este trabajo no requiere definir nuevos perfiles, utiliza uno recientemente estandarizado e introduce una leve variación semántica en el metamodelo del diagrama de clases y del diagrama de expresiones de VSL.
- En sistemas donde una operación participe de distintos escenarios, sea el sistema distribuido o no, es posible especificar diferentes valores para la misma propiedad de la operación. Si el sistema no es distribuido, se selecciona el valor de la propiedad que satisfaga todos los requisitos y se implementa la operación, o se utiliza polimorfismo si lo que se desea es optimizar otras características. Si el sistema es distribuido, se pueden implementar diferentes operaciones, esto de acuerdo al procesador que lo ejecute.

7.2. Trabajos futuros

Existen muchas posibles líneas de trabajo que están aún sin explorar, o que han sido exploradas parcialmente. Entre ellas están:

- Explorar nuevas relaciones semánticas entre los elementos del modelo y el metamodelo que sean fruto de la propuesta aquí planteada.
- Especificar estructuralmente otros elementos de un escenario diferentes a las operaciones: componentes, servicios Web, etc.
- Utilizando UML, MARTE y la especificación de NFRs en el diagrama de clases, se requiere analizar el desempeño de sistemas embebidos empleando técnicas tradicionales a partir de diagramas de clase UML.
- Especificar otro tipo de NFRs que no sean temporales, esto es, recursos hardware como memoria, consumo de energía, seguridad, entre otros.