Especificación de Requisitos de Desempeño en el Diagrama de Clases

Sergio Serna
Centro de Investigación
Instituto Tecnológico Metropolitano
Cra 31 # 54 - 10 Medellín, Colombia
Email: sergioserna@itm.edu.co

Fernando Arango
Escuela de Sistemas
Facultad de Minas, Universidad Nacional
Cra 80 # 65 - 223 Medellín, Colombia
Email: farango@unal.edu.co

Resumen— La especificación de requisitos no funcionales en sistemas software, ha presentado múltiples retos a académicos investigadores interesados en el tema. Las cada vez más crecientes exigencias de los sistemas en atributos catalogados, dentro de los que se denominan requisitos no funcionales, como desempeño, seguridad, escalabilidad, entre otros, ha permitido diversos enfoques a la hora de construir los planos software del sistema deseado por el cliente. Este trabajo se enmarca dentro de la especificación formal de requisitos temporales no funcionales, utilizando el lenguaje de modelado unificado para la construcción de los planos software. El trabajo no trata sobre requisitos funcionales o no funcionales que no están relacionados con tiempo. La especificación se hace sólo sobre diagramas de clase UML, y 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 del modelo con los del metamodelo, logrando una mayor expresividad y permitiendo tomar decisiones de implementación que antes no era posible en esta etapa del desarrollo. Para lograr esto, es necesario realizar una variante a la semántica del diagrama de clases, permitiendo relacionar metaclases que antes no estaban relacionadas. Igualmente, se introduce una nueva simbología para expresar la nueva metarelación presente en el diagrama de clases.

Palabras clave— Diagrama de clases, Metamodelo, Requisitos no funcionales, UML.

Abstract— Specification of non-functional requirements in software systems represents a challenge to academic researchers interested in this subject. The increasing demand for catalogued attributes, among which are the non-functional requirements (performance, security, scalability and others) has given rise to different approaches to building the software for the system required by the customer. This work deals with the specification of non-functional requirements of timing, using a unified modeling language for building the software plans. The specification is made only on UML class diagrams, and it is supported on existing methods at the early stages of development, in order to elicit requirements for consistency of the diagrams, and reaching a new class diagram. This new class diagram relates elements of the model and metamodel, to achieve a greater expressiveness and enabling decisions for implementation, which formerly were not possible at this stage of development. To achieve this, it was necessary to perform a variant on the semantics of the class diagram. This allowed relating metaclasses that were not related before. A new symbology is also introduced, in order to express the new metarelation that is present in the class diagram.

Keywords— Class diagram, metamodel, non-functional requirements, UML.

INTRODUCCIÓN

El éxito de un sistema software se mide principalmente por el grado en el cual se cumplen 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. Por esto, es indispensable tener una base de requisitos estable antes de comenzar el proyecto, si no es así, lo más probable es que se fracase.

La ingeniería de requisitos (RE - Requirements Engineering) [1] [2] se ha enfocado, fundamentalmente, en la elicitación de los llamados requisitos funcionales (FRs – Functional Requirements) [3], mientras que los requisitos no funcionales (NFRs - Non-Functional Requirements) se han tratado de una forma distinta [4] [5] [6] [7].

Los NFRs son requisitos muy generales, que imponen restricciones a las soluciones de acuerdo a las posibilidades del mundo real. Por ejemplo, una red de sensores puede requerir que todos sus nodos sean consultados una vez cada cinco segundo. Este requisito es transversal a muchos artefactos del sistema, y de alguna manera restringe la plataforma de ejecución.

Mientras que los FRs describen que debe el sistema o dispositivo hacer, los NFRs se concentran en la manera como el sistema o dispositivo debe acompañarse de tal forma que se satisfagan los FRs [8]. 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 las 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 desempeñar.
- 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.

Hay una gran variedad de NFRs, los cuales es indispensable tener en cuenta ya que generalmente pueden ser restricciones al proceso de desarrollo, al producto o presentarse como requisitos externos. Sin embargo, debe tenerse en cuenta 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. Por lo tanto, basado en [4] y [9], se pueden categorizar los NFRs en restricciones de interfaz, de desempeño, operativas, de recursos, de verificación, de documentación, de portabilidad, de seguridad, de calidad, entre otras.

Nuestro interés se centra en los requisitos de desempeño, los cuales incluyen, entre otros, propiedades de tiempo y espacio.

A. Desempeño

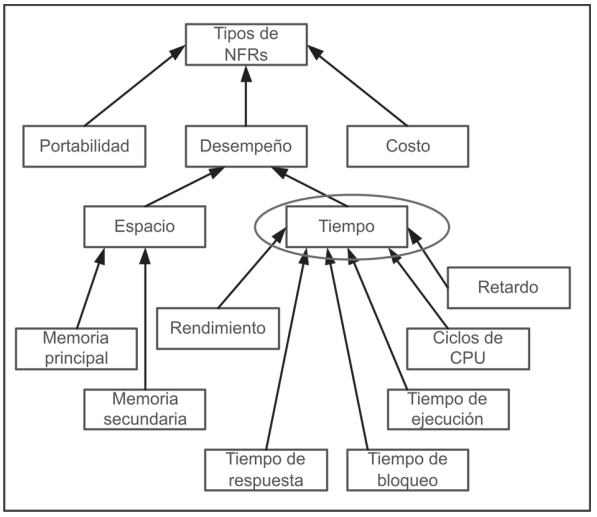
Como con la mayoría de los NFRs, tratar con requisitos de desempeño no es fácil, ya que se hace necesario resolver los conflictos resultantes de la interacción entre diferentes NFRs y entre sus múltiples implementaciones. Para especificar el desempeño de un sistema, es necesario tener claridad acerca de lo que significa buen desempeño [10]: 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 categorías 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 (calidad de servicio).

La dificultad de tratar con requisitos de desempeño, radica básicamente en que estos requisitos tienen un impacto global sobre todo 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 subdivididos en conceptos más especializados (véase Fig. 1), que finalmente pueden ser operacionalizables [4].

Fig. 1. TIPOS DE NFRS



Para lograr el buen desempeño de un sistema software, es necesario enfocarse en los elementos que son prioridad y tratar de compensar aquellos aspectos que no pueden satisfacerse completamente. En la literatura, existen varias metodologías de trabajo para construir sistemas con buen desempeño, los cuales, dependen fundamentalmente del dominio del sistema que se esté tratando: para cadenas de suministro [11], para sistemas cliente/servidor [17], para sistemas distribuidos [13], para sistemas de información [4] [14], para protocolos de transporte [15], entre otros.

II. ESPECIFICACIÓN DEL DESEMPEÑO

La especificación de NFRs se ha realizado de diversas formas. Uno de los trabajos más destacados tiene que ver con un método gráfico cuyo eje central es el concepto de softgoal, el cual define un marco de trabajo basado en NFRs [4].

Otros métodos de especificación de NFRs utilizan ontologías, con estas logran establecer, en algunos casos, relaciones semánticas entre el contexto y los NFRs [7] [16]. En otros casos, los NFRs se expresan independientemente del dominio de aplicación, aunque se asocien con requisitos de calidad del servicio – QoS [17]. En el mapeo de NFRs directa y explícitamente desde el proceso de ingeniería de requisitos hasta el diseño de la arquitectura, se destacan los trabajos realizados por [5] [18]. Otras técnicas definen las propiedades no funcionales relacionadas con desempeño en el marco de UML (Unified Modeling Language) y OCL (Object Constraint Language) [19] [20] [21] [22] [23] [24].

El proceso de especificación de los requisitos de desempeño, es una combinación de métodos de elicitación de requisitos tradicionales [3] y métodos de elicitación de NFRs [4]; posteriormente, se utiliza el perfil UML para sistemas embebidos de tiempo real [19]. Finalmente se traslada esta especificación al diagrama de clases [25].

A. Elicitación de requisitos

El proceso de elicitación de requisitos se realiza utilizando UN-METODO [3]. Este método tiene cuatro grandes hitos: contexto del software, análisis del problema, propuestas de solución y esquema conceptual. En el contexto del software se conoce el marco organizacional, los objetivos del área de aplicación, la organización del área, los actores e interesados, los roles de los actores y el vocabulario propio del área. Esta fase produce los actores, el esquema preconceptual y el modelo del dominio.

El análisis del problema, trata de describir los procesos propios del área: qué hacen, quién los lleva a cabo, qué información involucran, cuánto duran, cuántas veces se llevan a cabo, dónde se llevan a cabo, qué regulaciones deben tener en cuenta, qué problemas tienen, en qué grado los procesos del área dan cuanta de los objetivos del área, qué problemas deben ser resueltos (para dar solución a los objetivos), y cuáles son sus causas. Esta fase produce el diagrama de procesos, la tabla explicativa de los procesos, el diccionario de datos, el diagrama de objetivos y el diagrama de causa-efecto. Debido al hecho de que este trabajo se enfoca en un tipo de NFR, desempeño, el diagrama de objetivos de esta fase es sustituido por una estructura más adecuada para trabajar con NFRs. Esta estructura sirve para representar y registrar el proceso de diseño y razonamiento en grafos, llamado grafos de interdependencia softgoal (SIG - Softgoal Interdependency Graphs) [4]. 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. Un softgoal, es un objetivo “soft” por naturaleza, o mejor, es un NFR que el sistema bajo desarrollo debe alcanzar. Este, no necesariamente tiene un criterio absolutamente claro de satisfacibilidad.

En la fase de propuestas de solución, se plantea la nueva estructura organizacional, los nuevos procesos, indicando qué hacen, quién los lleva a cabo, qué información involucran, cuánto duran, cuántas veces se llevan a cabo, dónde se llevan a cabo, qué regulaciones se deben tener en cuenta, qué problemas tienen, entre otros; igualmente se determina la manera como participa el sistema informático en los nuevos procesos, los efectos de la nueva forma de proceder sobre los problemas y objetivos del área, el costo aproximado, y los factores críticos del éxito de la nueva propuesta. Como resultado de esta fase se tiene un nuevo organigrama, un cambio en los actores, un nuevo diagrama de procesos, el diagrama de casos de uso, una carta de navegación de interfaces, la valoración y costeo de la propuesta de solución y los factores críticos de éxito de la solución.

El esquema conceptual trata de identificar los objetos del sistema, los aspectos estructurales y dinámicos de los objetos del sistema y las restricciones sobre los objetos del sistema que son necesarias para garantizar el comportamiento propuesto. Esta fase entrega como resultado las consultas y transacciones, el diagrama de clases, las derivaciones y restricciones, y los eventos y operaciones.

Posterior al proceso de elicitación de requisitos, viene la especificación, en diagramas UML, de los aspectos relevantes al tiempo y al desempeño.

B. Especificación de aspectos de desempeño en diagramas UML

Existen múltiples trabajos que intentan modelar desempeño con UML, entre ellos están [19] [20] [21] [22] [23] y [24]. En esta sección trabajaremos con MARTE, ya que es una especificación estándar muy reciente.

El perfil UML para análisis y modelado de sistemas embebidos de tiempo real (MARTE - Modeling and Analysis of Real-Time Embedded Systems) [19], permite especificar características de desempeño y planificación, que facilitan el modelado y posterior análisis de algunos tipos de NFRs. 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[1], 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. Está compuesto por un conjunto de suboperaciones 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, 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.

Otro elemento clave son los recursos - Resource, los cuales 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 protegidos. 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 físico o virtual, capaz 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.

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 [19], 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 [27] [26].

La especificación del desempeño deseado de un sistema, debe permitir estimar el desempeño de una instancia del sistema, de acuerdo con el modelo para él planteado, e igualmente, debe facilitar la mejora del sistema, identificando cuellos de

[1] Se utilizarán los nombre sin traducir al castellano, ya que una traducción generaría confusión con la especificación actual

botella y recursos críticos. Para esto, es necesario analizar el sistema bajo diferentes escenarios, con parámetros diferentes cada vez, pero manteniendo la estructura general del mismo.

El modelo del dominio para analizar el desempeño de un sistema (véase Fig.2), está basado en el modelo de análisis cuantitativo genérico (Generic Quantitative Analysis Modeling - GQAM)2 [19]. El núcleo del dominio del GQAM, tiene como eje fundamental la descripción de como el sistema, mientras realiza su función, utiliza los recursos.

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 requerimientos que se le hagan, sino también en forma oportuna. Igualmente, debe estar en capacidad de atender requerimientos 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. 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 I), 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.

[2] 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

Fig. 2. MODELO DEL DOMINIO PARA ANÁLISIS DE DESEMPEÑO

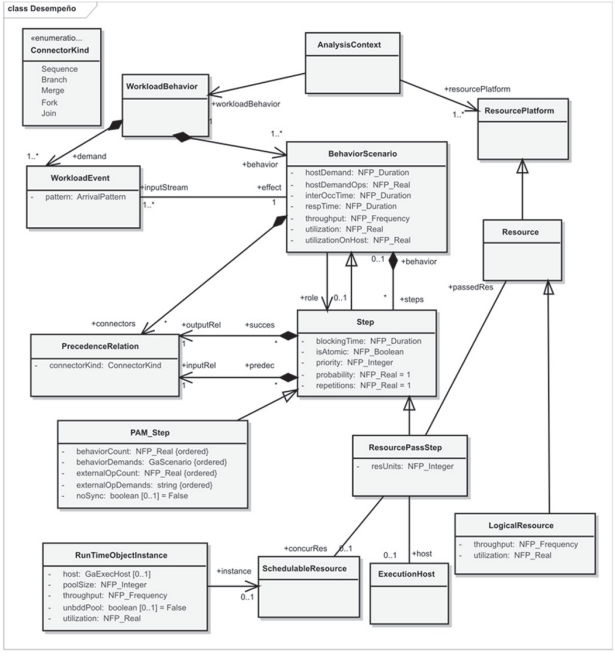


Tabla I. Propiedades útiles para analizar el desempeño

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 ejecutarse su predecesor
hostDemand: NFP Duration[*]	La demanda de CPU 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 escenario 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

La especificación UML para analizar el desempeño de un sistema software [19], lo que deja claro con respecto a BehaviorScenario y Step, es que un conjunto secuencial de Step componen un BehaviorScenario. De hecho, un Step también 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.

III. REPRESENTACIÓN DE NFRS EN EL DIAGRAMA DE CLASES

Las clases tienen asociadas propiedades y operaciones, y cada una de estas está relacionada con otros elementos del metamodelo del diagrama de clases. Pero en ninguna parte del metamodelo se establece una relación entre las operaciones y las propiedades. Este trabajo propone una relación del tipo composición entre las metaclases property y operation en el marco del metamodelo del diagrama de clases (véase Fig. 3). Con esta propuesta las operaciones de las clases tendrán atributos propios, los cuales permitirán representar propiedades como el tiempo de ejecución de un método.

La relación es de composición, donde la propiedad no puede existir sin la operación. Los extremos de la relación son:

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

Cualquier propiedad definida en MARTE [19] a la que se le permita mapearse a métodos de una clase, puede, con esta nueva propuesta semántica, representarse en el diagrama de clases del sistema.

La nueva sintaxis para representar clases, métodos y atributos de los métodos en el diagrama de clases puede verse en la Fig. 4.

Nótese la relación entre el método y la clase, es una composición en un tono gris y de línea más gruesa que la composición regular. Esto se debe fundamentalmente al hecho de que esta ya no es una relación entre dos clases del modelo, si no más bien una metarelación entre el método y la clase a la cual pertenece.

Con esta nueva sintaxis, se está creando un nuevo diagrama de clases que permite combinar elementos del modelo con elementos del metamodelo. Esta representación le da mayor expresividad al modelo, permitiendo conocer elementos que antes de esta forma de especificar NFRs no se podían inferir de uno de los diagramas estructurales más importantes de un modelo UML, el diagrama de clases.

IV. 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.

Fig. 3. PROPUESTA DE MODIFICACIÓN AL METAMODELO DEL DIAGRAMA DE CLASES

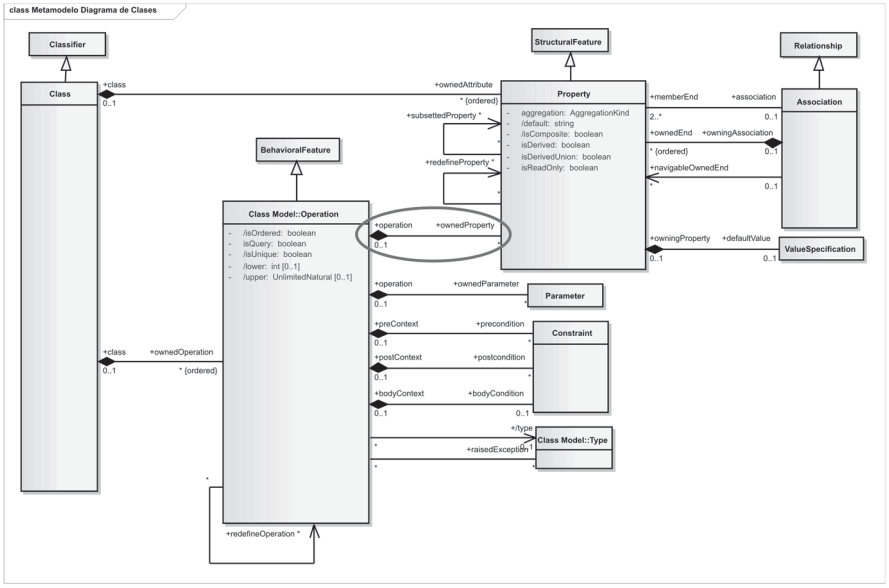
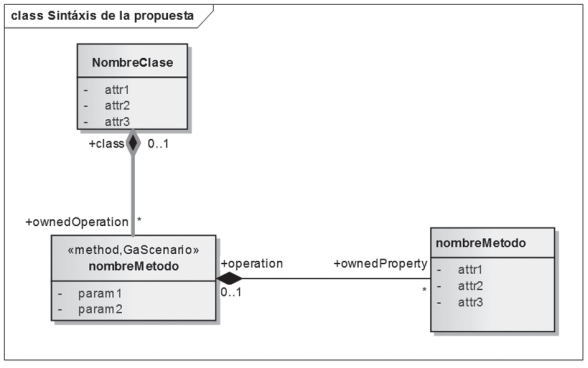


Fig. 4. SINTAXIS PARA LA REPRESENTACIÓN DE ATRIBUTOS DE UN MÉTODO



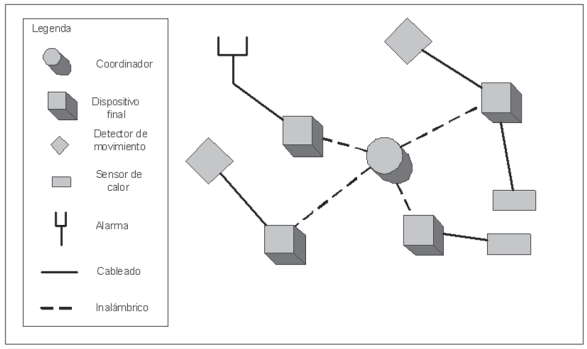
El sistema de seguridad es un sistema de tiempo real, ya que tiene varias exigencias asociadas con el tiempo que se enuncian a continuación:

- 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 500 ms.

Por cuestiones de espacio, la especificación la comenzaremos a mostrar desde el momento en que empezamos a utilizar el marco de trabajo basado en NFRs [4], el cual nos permite especificar el sistema a partir de sus NFRs.

La topología propuesta para la WPAN puede verse en la Fig. 5. Aquí se distribuye la red entre un nodo coordinador, y cuatro dispositivos finales; tres de los dispositivos finales se utilizan 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.

Fig. 5. TOPOLOGÍA DE LA WPAN



Partiendo del softgoal seguridad en el campamento, Seguridad[campamento], se van indicando los demás softgoal que los subrogan (véase Fig. 6a). Para este caso se tienen seguridad en los sectores, Seguridad[sector], confiabilidad del coordinador, Confiabilidad[coordinador], desempeño de la WPAN, Desempeño[wpan] y disponibilidad de la alarma, Disponibilidad[alarm]. De acuerdo a los demás softgoals presentes, se puede determinar el grado de satisfacibilidad de cada softgoal, el cual se representa con una nube. Nótese que el 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 + en el enlace indica que el softgoal fuente aporta positivamente a la realización del softgoal destino, mientras que un signo - es todo lo contrario.

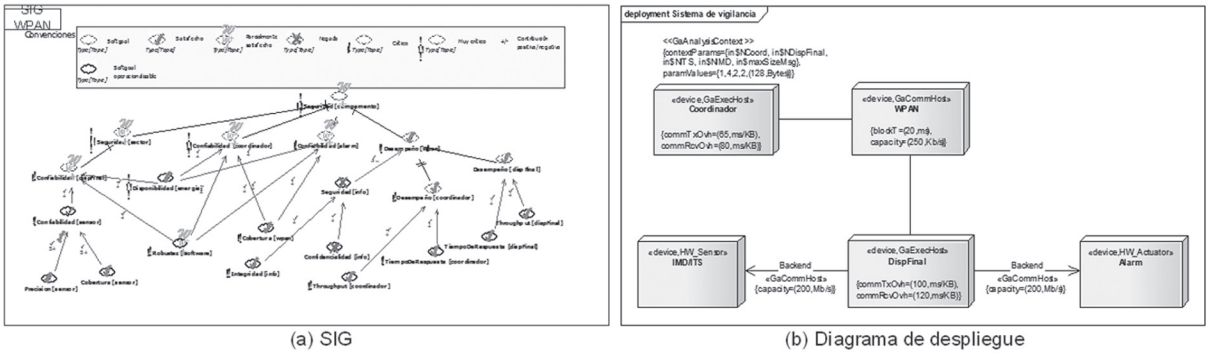
Los softgoals leer estado de un sector, LeerValor[sector], y leer estado de un sensor, LeerValor[sensor], son el punto final de la especificación en este SIG (Softgoal Interdependency Graphs). A partir de acá, se empieza a modelar, en un lenguaje como UML, con el fin de obtener los planos que permitan la construcción del sistema que cumpla los NFRs requeridos por el cliente.

El diagrama de despliegue del sistema puede verse en la Fig. 6b. Aquí se observa que el contexto de desempeño del sistema hace énfasis, utilizando el lenguaje de especificación de valores (Value Specification Language - VSL) [19], 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 <<GaAnalysisContext>> reúne la especificación del contexto de desempeño de la siguiente forma:

<<GaAnalysisContext >>

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

Fig. 6. DIAGRAMAS QUE ESPECIFICAN LA WPAN



(a) SIG (b) Diagrama de despliegue

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>>. <<GaExecHost>> representa 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 [28], el canal de comunicación, por limitaciones físicas del estándar [29], tiene una capacidad de 250 Kb/s. La latencia de la WPAN es típicamente del orden de los 20 ms.

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 200 Mb/s.

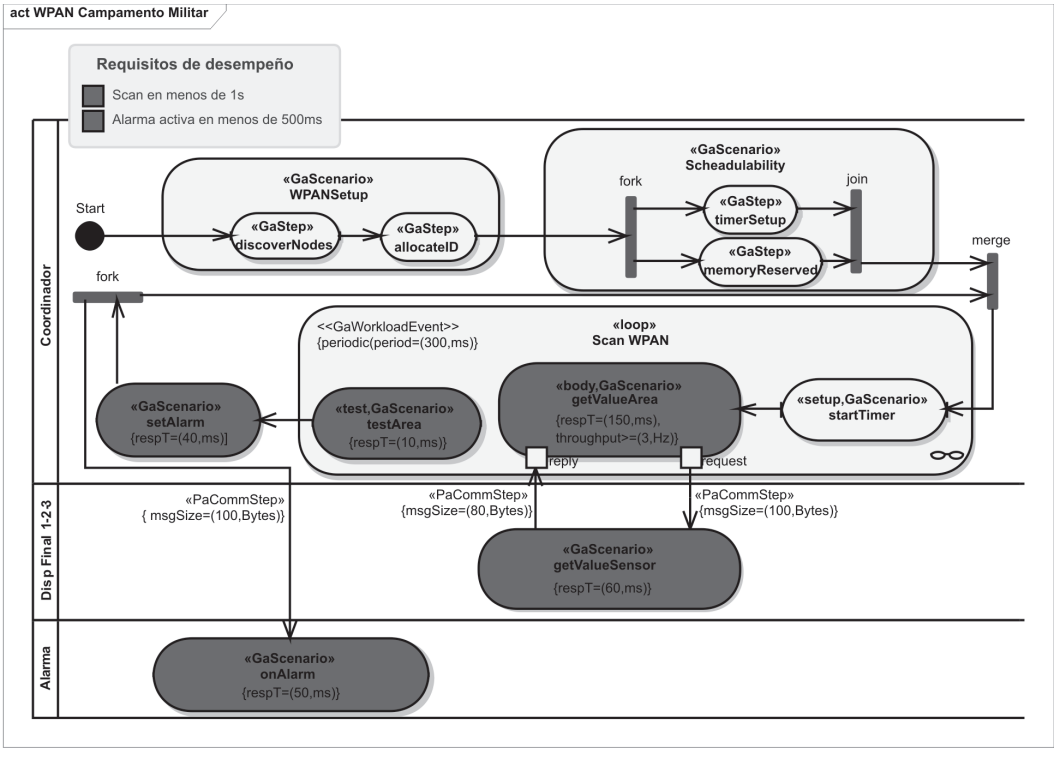
Sobre los nodos se despliegan los diferentes artefactos que se encargarán de los hilos del sistema. El coordinador y los dispositivos finales deben tener artefactos para llevar información al buffer de memoria, almacenar la información de los sensores y el estado de los sectores, y administrar los recursos lógicos dados por el pool de buffers.

El coordinador debe contar con un artefacto que le permite realizar las consultas cada segundo del estado de los dispositivos finales, mientras que a los dispositivos finales les permite leer el estado de los sensores. Por otra parte, debe existir un artefacto que le permiten a los dispositivos finales avisar oportunamente al coordinador la presencia de intrusos en su zona, mientras que al coordinador le permite enviar la orden de activar alarma al dispositivo final encargado de ello.

La planificación del proceso de control de toda la WPAN debe llevarla a cabo un artefacto software residente en el coordinador, y que regularmente es administrado por el sistema operativo.

El diagrama de actividades (véase Fig. 7) detalla mucho más los requerimientos de tiempo y recursos que necesita la WPAN para cumplir su función. Inicialmente se identifican los responsables de cada actividad: coordinador, dispositivo final 1-2-3 y alarma.

Fig. 7. DIAGRAMA DE ACTIVIDADES DE LA WPAN



Existe un GaWorkloadEvent, el cual garantiza el sensado de todos los dispositivos finales de la WPAN cada segundo, ya que dispara un nuevo evento iniciador 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 si-

guiente sector, no deben tardar más que el periodo del GaWorkloadEvent.

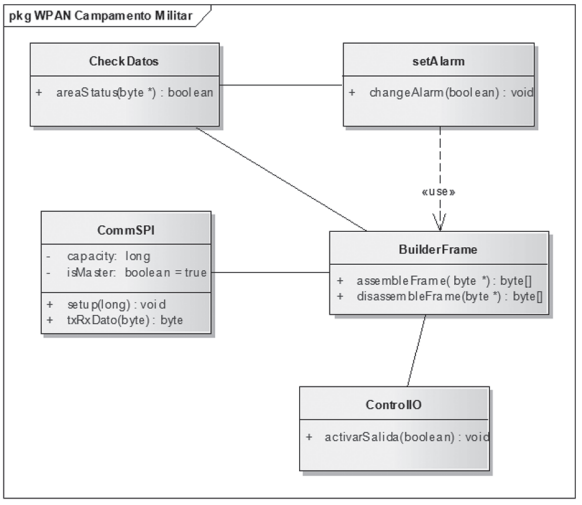
Las actividades en este diagrama están estereotipadas con GaScenario, de esta forma, es posible asociarles atributos definidos en MARTE [19], y que son útiles para analizar y modelar desempeño. Los escenarios resaltados con rojo, son aquellos que se ejecutarán periódicamente mientras no se detecte un intruso en uno de los sectores del campamento militar. Estos escenarios tienen un respT, definido en la Tabla I, 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 de getValueArea están los 60 ms de tiempo de respuesta de getValueSensor.De esta forma, getValueArea debe tener un tiempo de respuesta máximo de 90 ms, incluido 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 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 incluido 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 clases que se obtiene para este sistema, puede verse en la Fig. 8. Allí se establecen las relaciones entre las clases, las propiedades y los métodos de cada una.

Fig. 8. DIAGRAMA DE CLASESDE LA WPAN



El diagrama de secuencias que se obtiene a partir del diagrama de actividades y el diagrama de clases, puede verse en la Fig. 9. Allí se siguen cumpliendo los NFRs temporales especificados por el cliente. En este diagrama ya no se asocian tiempos a actividades, si no que cada actividad se descompone en un conjunto de secuencias de métodos, los cuales tienen diferentes requerimientos en cuanto a tiempos de ejecución. Se puede hacer seguimiento a cada NFR, según el color de los mensajes.

El diagrama de secuencias nos permite establecer diferentes tiempos de respuesta para cada método, 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 requerimientos 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, esto es,

$$respT = \min(respT_0, respT_1, \ldots, respT_{n-1})$$

En la Fig. 8 se observa el diagrama de clases del sistema sin introducir la propuesta presentada en este trabajo. Allí se observan las clases, sus atributos y métodos, y sus relaciones con otras clases. En la Fig. 10 se muestra el nuevo diagrama de clases del sistema de vigilancia para un campamento militar basado en una WPAN. Este diagrama de clases es más grande que el diagrama de clases original, debido a la nueva representación de las propiedades de los métodos de una clase. No a todos los métodos se les representó sus propiedades, sólo se tomaron aquellos métodos de aquellas clases que intervienen en el NFR temporal que se desea satisfacer.

Son múltiples las NFPs que pueden representarse para un método utilizando esta propuesta, pero este sistema de vigilancia sólo requiere representar dos propiedades a sus métodos: el tiempo de respuesta, respT, y el host donde se ejecutará el método, host. La propiedad host, permite establecer una relación directa entre el método y el host que lo ejecutará, de tal forma que se puedan tomar decisiones de implementación en esta fase del desarrollo del proyecto, ya que para el mismo método se pueden seleccionar diferentes plataformas de ejecución, diferentes lenguajes de programas o diferentes enfoques para la solución de la función del método. Por ejemplo, el método assembleFrame de la clase BuilderFrame, tiene diferentes requerimientos en cuanto al tiempo de respuesta, ya que en el host coordinador se le exigen 5 ms, mientras que en el host dispositivo final la exigencia son 8 ms.

V. CONCLUSIONES Y TRABAJOS FUTUROS

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 al método. 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 las 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. Se establece una relación de composición, hasta ahora inexistente, entre las metaclases property y operation.
- Para la notación se utiliza un símbolo de uso corriente en los diagramas de clase, la asociación de composición. Con este símbolo, matizado por un tono gris y una línea más gruesa, se puede indicar la metarelación entre el método y la clase a la cual pertenece.
- No se introducen nuevos elementos al metamodelo, es suficiente con los que se tiene en el diagrama de clases, solo se requiere 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.

Fig. 9. DIAGRAMA DE SECUENCIAS DE LA WPAN

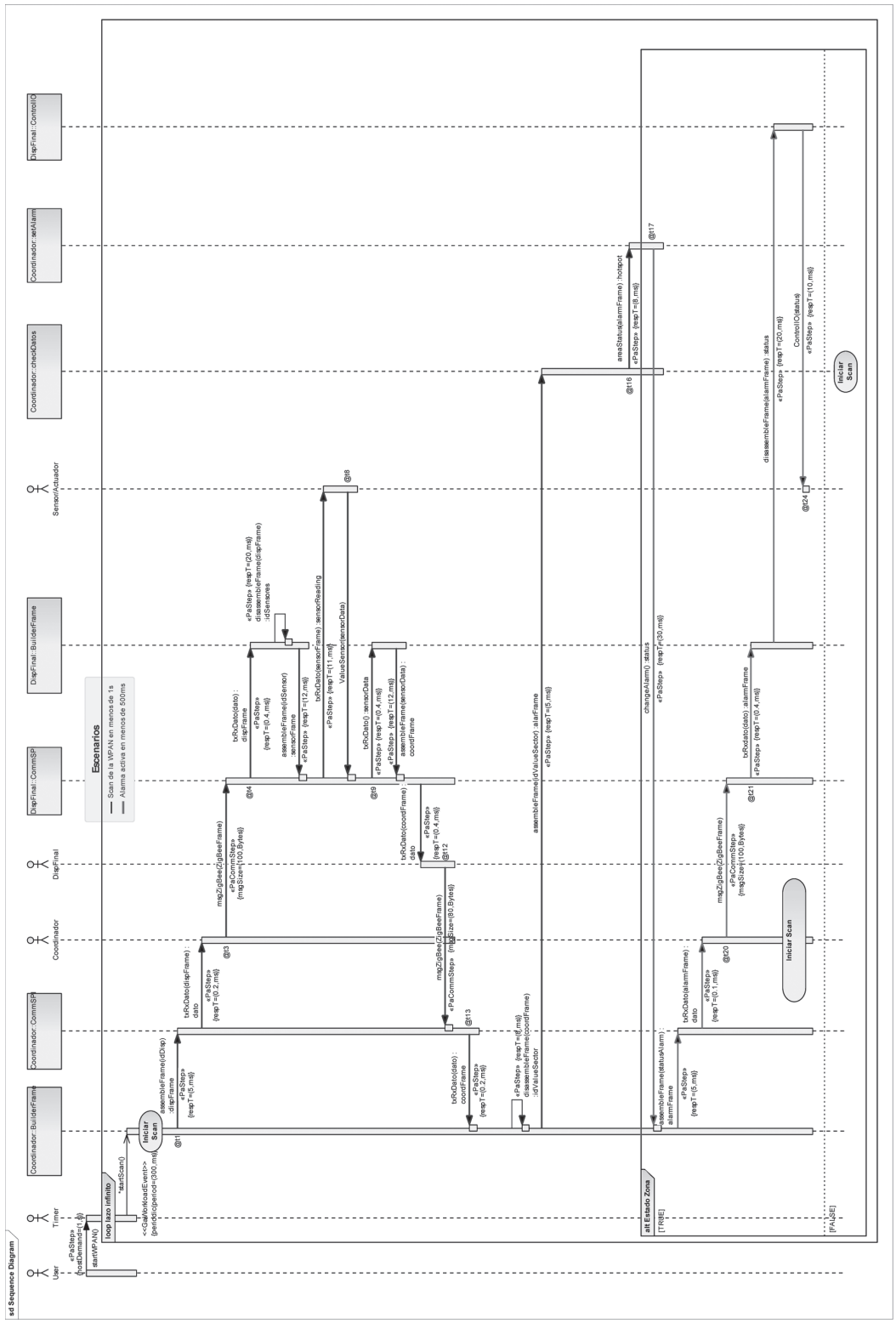
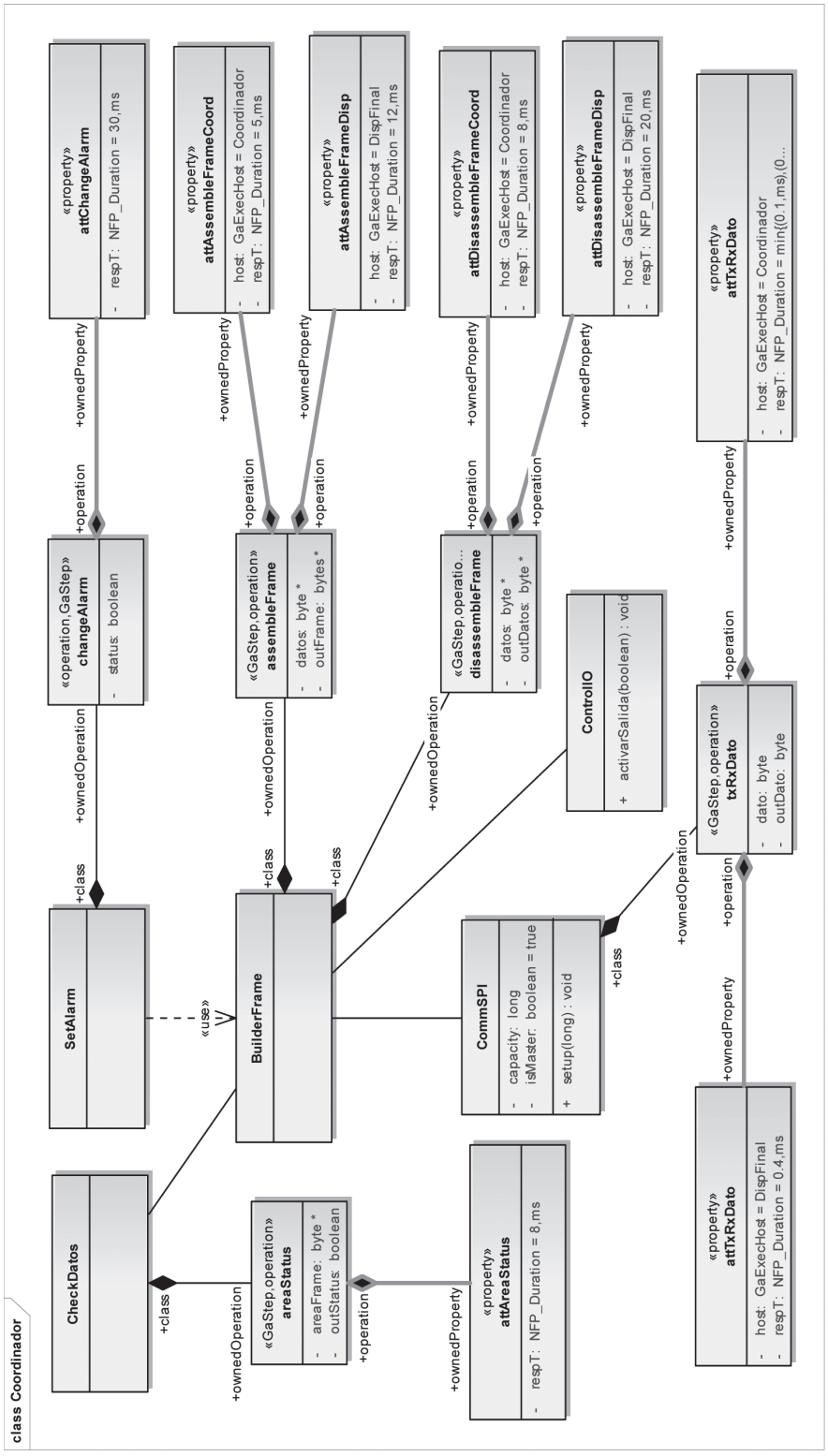


Fig. 10. DIAGRAMA DE CLASES DE LA WPAN CON LA NUEVA PROPUESTA



- Las especificaciones cuantitativas pero informales de ciertos tipos de requisitos temporales no funcionales dadas por el cliente, pueden desagregarse en escenarios y métodos que posteriormente se utilicen para garantizar 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 los métodos son explícitas en el diagrama de clases, y se corresponden con los requisitos temporales no funcionales especificados en la etapa de elicitación 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.
- En sistemas donde un método participe de distintos escenarios, sea el sistema distribuido o no, es posible especificar diferentes valores para la misma propiedad del método. Si el sistema no es distribuido, se selecciona el valor de la propiedad que satisfaga todos los requisitos y se implementa el método, o se utiliza polimorfismo si lo que se desea es optimizar otras características. Si el sistema es distribuido, se pueden implementar diferentes métodos, esto de acuerdo al procesador que lo ejecute.
- Existen muchas líneas de trabajo aun 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 los métodos: componentes y servicios Web, entre otros.
- 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.

Agradecimientos

Este trabajo ha contado con el apoyo decidido del Instituto Tecnológico Metropolitano y la Universidad Nacional de Colombia.

REFERENCIAS

[1] E. Hull, K. Jackson, and J. Dick, Requirements Engineering, 2nd ed. Springer, 2005

[2] B. Nuseibeh and S. Easterbrook, “Requirements engineering: a roadmap,” in ICSE ’00: Proceedings of the Conference on The Future of Software Engineering. New York, NY, USA: ACM, 2000, pp. 35–46

[3] F. Arango and C. M. Zapata, UN-Método para la elicitación de requisitos de software, C. M. Zapata, Ed. Escuela de Sistemas Universidad Nacional de Colombia, Medellín, 2006

[4] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, Nonfunctional requirements in software engineering. Kluwer Academic Publishers Group, 2000

[5] L. Xu, H. Ziv, and D. Richardson, “Towards modeling non-functional requirements in software architecture,” in Early Aspects 2005: Aspect-Oriented Requirements Engineering and Architecture Design, in conjunction with AOSD’2005, Chicago, Illinois, USA, March 2005

[6] P. Botella, X. Burgues, X. Franch, M. Huerta, and G. Salazar, “Modelling non-functional requirements,” in Proceedings of Jornadas Ingeniera de Requisitos Aplicados (JIRA), Sevilla, Spain, 2001

[7] M. Dinkel and U. Baumgarten, “Modeling nonfunctional requirements: a basis for dynamic systems management,” in SEAS ’05: Proceedings of the second international workshop on Software engineering for automotive systems. New York, NY, USA: ACM, 2005, pp. 1–8

[8] J. Burge and D. Brown, “Nfrs: Fact or fiction?” Computer Science Technical Report Worcester Polytechnic University, Tech. Rep. WPICS-TR-02-01, 2002

[9] IEEE, “IEEE recommended practice for software requirements specifications,” Tech. Rep., 1998

[10] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, Quantitative system performance: computer system analysis using queueing network models. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1984

[11] Q. Wang, J.-J. Yang, Y.-Z. Xu, and C.-X. Xing, "A method for semantic performance modeling in extended enterprise," in Industrial Engineering and Engineering Management, 2009. IE&EM '09. 16th International Conference on, Oct. 2009, pp. 1437–1441

[12] D. Menasce and H. Gomaa, "A method for design and performance modeling of client/server systems," Software Engineering, IEEE Transactions on, vol. 26, no. 11, pp. 1066–1085, Nov 2000

[13] A. Bagchi, "A study on managing the performance requirements of a distributed service delivery software system," Information and Software Technology, vol. 47, no. 11, pp. 735 – 746, 2005

[14] B. Nixon, "Management of performance requirements for information systems," Software Engineering, IEEE Transactions on, vol. 26, no. 12, pp. 1122–1146, Dec 2000

[15] S. Kiesel and M. Scharf, "Modeling and performance evaluation of transport protocols for firewall control," Computer Networks, vol. 51, no. 11, pp. 3232 – 3251, 2007

[16] T. Jingbai, H. Keqing, W. Chong, and L. Wei, "A context awareness non-functional requirements metamodel based on domain ontology," in Semantic Computing and Systems, 2008. WSCS '08. IEEE International Workshop on, July 2008, pp. 1–7

[17] G. Dobson, S. Hall, and G. Kotonya, "A domain-independent ontology for non-functional requirements," in e-Business Engineering, 2007. ICEBE 2007. IEEE International Conference on, Oct. 2007, pp. 563–566

[18] J. Wang, Y.-T. Song, and L. Chung, "From software architecture to design patterns: a case study of an nfr approach," in Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, 2005 and First ACIS International Workshop on Self-Assembling Wireless Networks. SNPD/SAWN 2005. Sixth International Conference on, May 2005, pp. 170–177

[19] Object Management Group, "UML profile for MARTE: Modeling and analysis of real-time embedded systems, version 1," Tech. Rep., 2009

[20] S. Graf, I. Ober, and I. Ober, "A real-time profile for uml," Int. J. Softw. Tools Technol. Transf., vol. 8, no. 2, pp. 113–127, 2006

[21] S. Flake and W. Mueller, "An ocl extension for real-time constraints," in Object Modeling with the OCL, The Rationale behind the Object Constraint Language. London, UK: Springer-Verlag, 2002, pp. 150 – 171

[22] ——, "A uml profile for real-time constraints with the ocl," in UML '02: Proceedings of the 5th International Conference on The Unified Modeling Language. London, UK: Springer-Verlag, 2002, pp. 179–195

[23] M. V. Cengarle and A. Knapp, "Towards ocl/rt," in FME '02: Proceedings of the International Symposium of Formal Methods Europe on Formal Methods - Getting IT Right. London, UK: Springer-Verlag, 2002, pp. 390 –409

[24] L. Lavazza, S. Morasca, and A. Morzenti, "A dual language approach to the development of time-critical systems," Electronic Notes in Theoretical Computer Science, vol. 116, pp. 227–239, 2005, proceedings of the International Workshop on Test and Analysis of Component Based Systems (TACoS 2004)

[25] S. Serna, "Especificación formal de requisitos temporales no funcionales," Master's thesis, Escuela de Sistemas, Facultad de Minas, Universidad Nacional de Colombia, 2010

[26] "IEEE std 1012 - 2004 ieee standard for software verification and validation," Tech. Rep., 2005

[27] "The standard performance evaluation corporation (spec)," http://www.spec.org/, January 2010

[28] ZigBee Alliance, "ZigBee Specification," Tech. Rep., January 2008. San Ramon, CA.

[29] "Part 15.4: Wireless medium access control (mac) and physical layer (phy) specifications for low-rate wireless personal area networks (wpans)," Tech. Rep., September 2006