UNIVERSIDAD NACIONAL DE SAN LUIS

FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS Y NATURALES

Tesis
para optar a la titulación de postgrado correspondiente a la
Maestría en Ingeniería de Software

Un Modelo Genérico para el Modelo de Negocio y su Formalización en Object-Z

Marcela Elena Daniele

Director: Lic. Gabriel Baum

San Luis
2007

Agradecimientos

Agradezco a todas las personas que de una u otra manera me ayudaron para lograr la concreción de este proyecto. Aunque son muchas, no puedo dejar de mencionar algunas de ellas. En primer lugar quiero agradecer a mi esposo Jorge y a mis hijos Bruno y Augusto, por su paciencia y contención afectiva. A mis padres, por ser incondicionales en todo, siempre. Agradezco a Ivana, que mucho más allá de ser nuestra secretaria, es mi gran amiga que siempre me ayuda y acompaña. Agradezco a Paola Martellotto por el apoyo que me brindó su trabajo en equipo y por confiar en mi propuesta. Gracias a Gabriel Baum por su tiempo y sus valiosos aportes.

Además, debo mencionar a la Universidad Nacional de Río Cuarto, y en particular a la Facultad de Ciencias de Exactas, Físico-Químicas y Naturales, donde trabajo desde hace más de 10 años, por darme el espacio y el apoyo económico que también hicieron posible la concreción de esta propuesta. A la Universidad Nacional de San Luis, que además de brindarme la posibilidad de optar a un título de posgraduación de excelente nivel, me permitió conocer muy buenas personas tanto en lo profesional como en lo personal.

Resumen

Cuando se inicia el desarrollo de un sistema de software, es fundamental que el ingeniero de software entienda la estructura y dinámica de la organización para la cual el sistema es realizado. Es necesario asegurar que todos los participantes del proyecto comprenden el funcionamiento del negocio involucrado y las mejoras que se esperan conseguir con el sistema informático que se desarrollará. Solo cuando el dominio del problema es completamente entendido, se puede proceder a la definición de las funcionalidades del sistema.

En este trabajo se presenta una técnica para construir modelos de negocio a partir de una reconstrucción de la propuesta original de la metodología de desarrollo de software Proceso Unificado [JBR99], y más específicamente de la propuesta de Rational Unified Process [RUP].

En este trabajo se muestra la definición un modelo genérico del modelo de negocio y un conjunto de reglas, que colaboran con el desarrollador de software en la creación de un modelo de negocio particular de una organización. Además, ayuda a determinar si un modelo de negocio previamente realizado se adecua al modelo genérico y a las reglas y mecanismos que este sugiere.

Índice

Figuras

CAPITULO 1: Introducción

1.1. Ingeniería de Software

Con el advenimiento de las computadoras de la 3era. Generación, comenzaron a producirse muchos problemas en el desarrollo de software, dado que no se contaba con técnicas y métodos que permitieran organizar y estructurar el desarrollo de sistemas de mayor tamaño y complejidad que los que se construían hasta ese momento. Hacia fines de los años 60 se produce la llamada “crisis del software”, caracterizada por síntomas tales como: carencia de fiabilidad, necesidad de mantenimiento permanente, retrasos en las entregas y costes superiores a los presupuestados, imposibilidad de mantener el código, falta de transparencia y pocas posibilidades reales de modificación o mejora.

A consecuencia de esta crisis surge la denominada “Ingeniería de Software”. Muchas fueron las discusiones para una definición apropiada de este término, pero se fue llegando al acuerdo común de que el desarrollo de software debe verse como el desarrollo de un producto de ingeniería que requiere planeamiento, análisis, diseño, implementación, prueba y mantenimiento [Pressman01], es decir, un trabajo de ingeniería, tal como lo ven los ingenieros de otras disciplinas que construyen sistemas complejos. En resumen, la Ingeniería del software es un proceso basado en metodologías o procedimientos, técnicas, teorías y herramientas que permite entender la naturaleza de un problema y dar una solución ingenieril al mismo. Los métodos o técnicas indican como construir técnicamente el software, las herramientas brindan un soporte automatizable para el proceso y los métodos, y el producto es el software que se realiza para un usuario o cliente con la documentación correspondiente. Los atributos del producto de software son las características mostradas por el producto cuando es instalado y puesto en uso. (Figura 1).

Una de las primeras definiciones dadas de Ingeniería de Software puede encontrarse en [Nau69], donde dice que este término indica *“el establecimiento y uso de principios de ingeniería para obtener económicamente software que sea fiable y que trabaje eficientemente sobre máquinas reales”.* Actualmente se conoce a la Ingeniería de Software como una disciplina dentro de las Ciencias de la Computación que se ocupa de la construcción de sistemas de software que debido a su gran tamaño y complejidad, deben ser construidos por un equipo de ingenieros [GJM91].

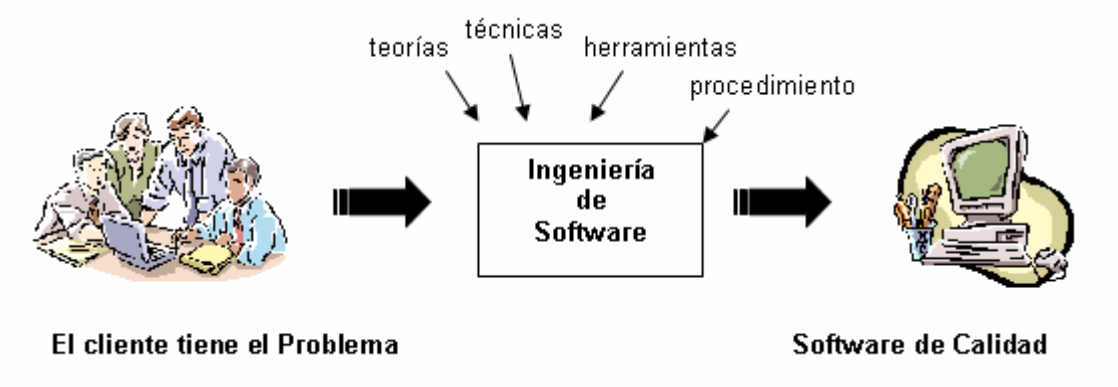


Figura 1: La Ingeniería de Software

En las primeras líneas del libro *Object Oriented Software Construction* [Meyer97] puede leerse: *"La ingeniería busca la calidad; la ingeniería de software es la producción de software de alta calidad"*, y expone el software de calidad como un conjunto de propiedades externas e internas que todo software debe cumplir. Las propiedades *externas* se asocian a los usuarios externos del sistema, y los *internas* solo son perceptibles por los profesionales que tienen acceso a las fuentes del software (por ejemplo: el sistema es modular, legible, estructurado). Las principales propiedades externas a tener en cuenta son:

- Corrección: capacidad del producto de software para ejecutar sus tareas de acuerdo a la definición dada en la especificación. Es necesario asegurar que cada capa (hardware, SO, compilador o aplicación) es correcta. Y cuando se construye software con reuso de librerías, primero se debe asegurar que cada librería es correcta.
- Robustez: capacidad del software de reaccionar apropiadamente a condiciones anormales. Complementa la corrección. Los casos normales y anormales son relativos a la especificación. Normal no significa deseable, significa planeado en el diseño del software. El rol de la robustez es asegurar que si ocurre un caso no esperado, el sistema no causará eventos catastróficos, dará mensajes de error apropiados y terminará su ejecución de manera "elegante".
- Extensibilidad: facilidad del software para adaptarse a los cambios de la especificación. Es un problema de escala. En programas pequeños es más fácil introducir cambios, a medida que el tamaño del programa crece los cambios son más difíciles de introducir y hasta pueden producir un colapso en todo el sistema. Dos principios indispensables para mejorar la extensibilidad: diseño simple y descentralización (módulos más autónomos).
- Reusabilidad: capacidad de los elementos de software de ser útiles para muchas aplicaciones diferentes. Los sistemas de software a menudo siguen patrones

similares, por lo tanto es necesario reducir el esfuerzo si hay algo que ya se resolvió antes.

- Compatibilidad: facilidad de los elementos de software de combinarse entre ellos. Formato de archivos estandarizados: por ej. en UNIX cada archivo de texto es simplemente una secuencia de caracteres; estructura de datos estandarizadas: por ej. en LISP, todos los datos y los programas son representados por árboles binarios, llamados “listas”; interfaces de usuario estandarizadas: por ej. en Windows, OS/2 y MacOS, donde todas las herramientas responden a un paradigma simple para la comunicación basada en ventanas, íconos, menues, etc.
- Eficiencia: capacidad de un sistema de software de hacer la menor cantidad de demandas posibles a los recursos de hardware, tales como el tiempo de procesador, la cantidad de memoria interna y externa.
- Portabilidad: facilidad de transferencia de productos de software a otros ambientes de software y hardware.
- Facilidad de uso: facilidad con la que las personas pueden usar un producto de software y aplicarlos para resolver sus problemas. Incluye facilidad de instalación y operación.
- Funcionalidad: extensión de la función, servicios, etc. que provee un sistema.
- Oportuno – A tiempo: disponible cuando los usuarios lo necesitan, o antes si es posible. Esta propiedad no cumplida es una de las grandes frustraciones de la industria del software.
- Verificabilidad: facilidad de preparar procedimientos de aceptación, especialmente test de datos, y procedimientos para detectar fallas.
- Integridad: proteger a sus componentes, ya sean programas o datos, de accesos y modificaciones no autorizadas.
- Reparabilidad: facilidad para reparar defectos.
- Economía: capacidad del equipo de desarrollo de completar el software de acuerdo al presupuesto pactado. Esta propiedad está muy relacionada con el tiempo de entrega del producto, y es otra frustración de la industria.
- Necesidad de documentación interna para los usuarios, documentación externa para los desarrolladores que necesitan comprender la estructura e implementación del sistema, y documentación de los módulos de interfaces para los desarrolladores que deben comprender la funcionalidad de un módulo sin necesidad de conocer su implementación.
- Capacidad de Mantenimiento: esta propiedad esta asociada a la capacidad del software para soportar diversos cambios sin producir un colapso en su funcionalidad. Los motivos que hacen cambiar un sistema son muy variados, pero

fundamentalmente pueden agruparse en: cambios en los requerimientos del usuario, cambios en los formatos de los datos, cambios en el hardware, corrección de errores, condiciones del mundo externo que provocan cambios en el software, etc.

Finalmente, los objetivos claves de la Ingeniería de Software se pueden resumir en: 1) contar con una metodología bien definida, dirigida a un ciclo de vida de planeamiento, desarrollo y mantenimiento; 2) documentar cada actividad en el ciclo de vida y 3) definir un conjunto de hitos predecibles que pueden ser revisados a intervalos regulares a través del ciclo de vida del software.

En la actualidad, se continúa estudiando y discutiendo la definición de procesos de producción de software que resulten apropiados para diversos tipos de proyectos. A pesar de los esfuerzos, existen aún sobrados casos de proyectos de software que fracasan.

Un error fundamental que a menudo cometen los desarrolladores de software es prestar poca atención a la tarea de estudiar detenidamente el dominio del problema y a partir de ello especificar las funcionalidades del sistema. A medida que los sistemas se tornan más complejos, la captura de los requerimientos se torna más importante y difícil de definir. Es allí donde surge la necesidad indiscutible del uso de metodologías de desarrollo de software y las técnicas de modelado, las cuales fueron creadas para simplificar la complejidad del desarrollo de sistemas.

1.2. El Proceso de Desarrollo de Software

Un proceso de desarrollo de software es un marco de trabajo de las actividades de ingeniería requeridas para construir un software de alta calidad. Un modelo de proceso de software es una estrategia de desarrollo que acompaña al proceso, técnicas y herramientas. Se selecciona un modelo de proceso para la ingeniería del software según la naturaleza del proyecto y de la aplicación, los métodos y herramientas a utilizarse, y los controles y entregas que se requieren.

En 1979, Tom DeMarco [DeMarco79] propone la Ingeniería de Software basada en modelos, comparando la construcción de un sistema de software con la de cualquier otro tipo de sistemas ingenieriles, y proponiendo la realización de modelos del sistema antes de la construcción del sistema mismo. De esta forma, el modelo de un sistema provee un medio de comunicación entre todos los participantes en el proyecto, cliente, usuarios y desarrolladores. La mayoría de los métodos de desarrollo de software que se utilizan en la actualidad adoptaron la filosofía propuesta por este autor y, salvando

las particularidades de cada uno, todos proponen la construcción de sistemas a partir de modelos previos.

Desde el punto de vista de la ingeniería de software, un modelo es una abstracción del sistema, especificándolo desde una “visión” determinada y un cierto nivel de abstracción. Los modelos que forman un sistema deben estar todos interrelacionados, y cada modelo construido se focaliza sobre el mundo real identificando, clasificando y abstrayendo los elementos que constituyen el problema y organizándolos en una estructura formal.

Toda aplicación o sistema implantado en una organización posee una arquitectura que describe su estructura y funciones. La idea es definir la estructura de los sistemas a través de modelos que describan la visión que tienen los distintos interesados de la organización para usarlos en la planificación y mejorar la toma de decisiones.

1.2.1. Modelos para el Desarrollo de Software

Un modelo de proceso o paradigma de ingeniería del software, se basa en la definición de una estrategia de desarrollo que guíe la construcción de artefactos de software. Según la naturaleza del proyecto y su aplicación, los métodos y herramientas a utilizarse, los controles y entregas a realizarse, se selecciona un modelo de proceso. A continuación se muestra un breve relato de métodos convencionales de la Ingeniería de software, principalmente extraídos del resumen que muestra Pressman en el capítulo 2 [Pressman01], que reflejan claramente dos enfoques bien diferenciados: el enfoque algorítmico, basado en la definición de procedimientos y funciones que se ejecutan sobre estructuras estáticas, y el enfoque orientado a objetos, basado en la definición de objetos como elementos pertenecientes al dominio del problema. Cada uno de los modelos exponen ventajas y desventajas, pero todos coinciden en un conjunto de fases genéricas: identificar requerimientos del usuario, analizarlos y generar un diseño en el lenguaje del desarrollador, implementar en un lenguaje de programación apropiado, y probarlo. Además de planificar tiempos de desarrollo, costos, plazos de entrega, herramientas a utilizar, recursos necesarios, etc.

1.2.1.1. El Modelo Secuencial Lineal

También llamado “ciclo de vida básico” o “modelo en cascada” sugiere un enfoque sistemático que involucra en forma lineal las siguientes actividades: análisis, diseño, codificación, pruebas y mantenimiento. A pesar de ser uno de los modelos más utilizados en la ingeniería de software, presenta varios problemas tales como: no siempre el proyecto se ajusta totalmente al modelo, es difícil que el usuario exponga los requerimientos completos al principio del proyecto, un error grave que aparece en

la etapa de programación puede ser fatal por lo avanzado que se encuentra el proyecto y el usuario no ve resultados por mucho tiempo.

1.2.1.2. El Modelo de Construcción de Prototipos

En este modelo el desarrollador y el cliente recolectan los requisitos, y a partir de ello se construye un diseño rápido y visible para ambos, a partir del cual se implementa un prototipo y es entregado al cliente con el único propósito de pueda evaluarlo y facilitar el refinamiento de los requisitos del software. Los inconvenientes que se generan son que a menudo el cliente no comprende que el prototipo ya cumplió su propósito y debe ser tirado, con la necesidad de desarrollar una versión del trabajo que cumpla con los requerimientos, las propiedades de calidad del software y la posibilidad de mantenimiento a largo plazo. Otro inconveniente que puede surgir, sobre todo con desarrolladores novatos, es que se asuman compromisos de implementación tan solo por hacer que el prototipo funcione lo más rápido posible.

1.2.1.3. El Modelo Incremental

Se trata de una combinación entre el modelo de desarrollo de software tradicional lineal, aplicado de manera repetida y el modelo de desarrollo a través de la construcción de prototipos. Cada secuencia lineal provoca un incremento del sistema y se realiza una entrega al cliente, este proceso se repite con cada incremento hasta la elaboración del producto completo. Desde una perspectiva de alto nivel, cada entrega puede ser vista como un prototipo. Sin embargo, a diferencia del método de construcción de prototipos clásico, cada entrega es un producto operativo.

1.2.1.4. El Modelo de Métodos Formales

Los métodos formales permiten que un ingeniero de software especifique, desarrolle y verifique un sistema aplicando una notación matemática y rigurosa. Los problemas tales como ambigüedad, incompletitud e inconsistencia se identifican y son corregidos con facilidad con la aplicación de métodos matemáticos.

Aunque aún no hay un enfoque establecido, los modelos de métodos formales prometen la realización de software libre de errores. No obstante, existen dificultades en su aplicación tales como: es un proceso caro y consume mucho tiempo, los desarrolladores deben ser profesionales expertos y dificultan la comunicación con el cliente. De todas maneras, en ciertas partes de los sistemas que requieren un alto nivel de seguridad muchos desarrolladores optan por estas metodologías.

1.2.1.5. Modelos de Procesos Evolutivos de Software

Una de las características que rigen los mercados actuales es la rapidez en la liberación de los productos. Debido a estas exigencias, las empresas están obligadas a entregar productos que implementen capacidades básicas iniciales, dejando abierto el desarrollo para que lanzamientos posteriores completen el sistema.

Cada una de las entregas futuras se planifican, por lo que los mecanismos que componen el proceso de desarrollo de software deben estar preparados para desarrollar (y ampliar) productos que evolucionen en el tiempo. Las actividades que constituyen el proceso de producción de software evolutivo pueden estar planificadas de diversos modos, y dan origen a distintas variantes de esta estrategia de desarrollo.

1.2.1.6. Modelo en Espiral

En este modelo, cada uno de los espacios de trabajos que componen el proceso de desarrollo lineal se ejecutan de manera iterativa, produciendo en cada una de las iteraciones, versiones cada vez más completas del sistema. Los espacios de trabajos o regiones de tareas típicos son: Comunicación con el Cliente, Planificación, Análisis de Riesgo, Ingeniería, Construcción y Adaptación, Evaluación del Cliente.

1.2.1.7. Modelo de Ensamblaje de Componentes

Esta estrategia de producción de software, pretende potenciar las características reusables de los componentes de software desarrollados, en general, con el paradigma Orientado a Objetos. Basados en los requisitos definidos por los usuarios, los componentes se seleccionan desde un repositorio de componentes reusables y se ensamblan para producir el sistema final. Este enfoque encierra el problema de la complejidad que implica la construcción y el mantenimiento de un repositorio de esta naturaleza. Los componentes una vez construidos tienen que ser catalogados, clasificados y almacenados siguiendo criterios bien definidos de organización, de modo que cuando sean requeridos durante el proceso de desarrollo, puedan ser identificados con relativa facilidad. También es necesario proveer todos los mecanismos que permitan establecer las diferencias semánticas entre componentes similares.

1.2.1.8. Modelo de Desarrollo Concurrente

Este modelo pretende capturar la visión según la cual, las actividades que integran el plan de proyecto, son realizadas simultáneamente por grupos de trabajos distintos. A su vez, cada una de estas actividades tienen un estado interno propio que las

caracteriza y la transición de un estado a otro dentro de una actividad, lo provoca algún acontecimiento específico.

1.2.1.9. El Proceso Unificado

El Proceso Unificado [JBR99] es una metodología de desarrollo de software configurable que se adapta a proyectos de variados tamaños y complejidad. Propone un conjunto de artefactos, organizados por etapas y fases, que sirven de guía para los participantes del proyecto. Se caracteriza por ser un modelo *dirigido por casos de uso*, *centrado en la arquitectura*, e *iterativo e incremental*. Los casos de uso son definidos en la primera etapa y dirigen la construcción de los artefactos en las etapas siguientes. Con este modelo, el trabajo es dividido en partes más pequeñas o miniproyectos, y cada miniproyecto es una iteración que resulta en un incremento al producto.

El Proceso Unificado utiliza UML [BRJ99] para representar los esquemas o artefactos de un sistema de software.

Para mayor detalle del Proceso Unificado ver el capítulo 3 y el anexo I, y acerca de UML en el capítulo 2.

1.3. Acerca de esta tesis

1.3.1. Motivación y Propuesta

Un proceso de desarrollo de software es un método que sirve para organizar las actividades de análisis, creación y mantenimiento de un producto de software. La definición de requisitos del sistema de software es la primera actividad definida por cualquier modelo de desarrollo de software. El Proceso Unificado es una metodología de desarrollo de software que sugiere para la primera etapa, la construcción de dos importantes modelos, el Modelo de Negocio y el Modelo de Casos de Uso del sistema. El primer modelo mencionado tiene por finalidad establecer una abstracción de la organización, y sobre la base de este conocimiento se construye el siguiente modelo para especificar los casos de uso y actores del sistema.

El Proceso Unificado de Rational, conocido como RUP (Rational Unified Process) [RUP], propone un workflow para definir el modelo de negocio y está conformado por un conjunto de artefactos. Se construyen dichos artefactos con el objeto de recolectar toda la información posible del negocio y permitir entender donde están los problemas, las necesidades de los clientes, la experiencia de los empleados, y las intenciones en general. Si se tiene en cuenta que una de las principales características de esta metodología indica que el sistema está *dirigido por casos de uso*, la construcción de un modelo de negocio apropiado colabora en la correcta definición de las

funcionalidades o casos de uso, y permite la obtención de un software de calidad, en tiempos y costos apropiados, redundando posteriormente en la construcción correcta del diseño, implementación y prueba del sistema.

Si el tamaño del negocio a modelar es pequeño, es fácil definir la construcción de los artefactos y organizar los resultados obtenidos. El inconveniente surge cuando se pretenden modelar negocios de mayor envergadura, y en cuyo caso es necesario contar con una manera sistemática para construir los artefactos necesarios y analizar los resultados obtenidos, con el fin de conseguir correctamente la definición de las funcionalidades del sistema.

En torno a esto, se realizaron numerosas lecturas de diversos trabajos (ver 1.3.2. Trabajos relacionados) que si bien utilizan la técnica de modelado de negocio propuesta por RUP, en ningún caso, se detectó una forma sistemática y simplificada que organice, relacione y priorice los artefactos a construir. Los trabajos mencionados describen diferentes técnicas en torno al uso del modelado de negocio, pero ninguno refleja claramente las relaciones entre los diecisiete artefactos propuestos por RUP, ni especifican reglas para verificar las instanciaciones realizadas.

El objetivo principal de este trabajo es facilitar al desarrollador de software el conocimiento, la comprensión y el desarrollo del contexto del sistema, a partir de las reglas y las relaciones definidas entre los artefactos que conforman el modelado de negocio, permitiendo construir un modelo suficiente y correcto para el conocimiento que se requiere del contexto. Además, puede colaborar con el análisis y control de un modelo de negocio construido previamente.

Sobre la base de un estudio detallado del modelo de negocio según la propuesta original del RUP [RUP], se propone un modelo genérico del modelo de negocio como una reconstrucción del modelo original, que permite visualizar con mayor claridad y especificar con mayor precisión los artefactos que componen el modelo de negocio y sus relaciones principales.

Mientras que el modelo original del RUP utiliza el lenguaje natural para describir los artefactos y sus relaciones, y esto lleva aparejados distintos riesgos de imprecisión, ambigüedad y dificultad de comunicación entre los miembros del proyecto, el modelo genérico propuesto simplifica la obtención de una solución al problema de modelado de negocio, ya que resume en un único diagrama de clases de UML todos los artefactos que componen el modelo de negocio, y las relaciones entre ellos. Además por la propia semántica del diagrama de clases de UML (tipo de relaciones, multiplicidad, navegabilidad, etc.) permite visualizar y especificar los artefactos necesarios e indispensables de construir y los opcionales.

El modelo genérico está formado por un diagrama de clases UML, que representa gráficamente los artefactos y sus relaciones, y un conjunto de reglas que completan el diagrama, definidas inicialmente en lenguaje natural.

Por último, con el objeto de asegurar que el modelo genérico propuesto pueda ser usado para expresar sin ambigüedad modelos de negocio concretos en dominios particulares, la propuesta es formalizada a través de una traducción completa de la semántica del modelo genérico del modelo de negocio al lenguaje formal Object-Z, facilitando considerablemente el análisis de las propiedades del sistema, mostrando posibles inconsistencias, ambigüedades o incompletitudes y dando una base sólida necesaria para la posible automatización del método.

1.3.2. Trabajos relacionados

Ninguno de los trabajos consultados ofrece un análisis de los artefactos propuestos por RUP [RUP] para el modelado del negocio. La mayoría se refiere a elementos en torno a los procesos de negocio y como construirlos, pero ninguno presenta una visión más general del modelo de negocio. Una breve reseña se expone a continuación:

En [OMMN01] se propone un método para obtener el modelo de requisitos a partir del modelo de negocio, donde solo utiliza algunos artefactos propuestos por el Proceso Unificado pero no justifica su selección. [Salm03] propone la utilización de las extensiones de UML propuestas por la OMG, para el Modelado de Negocio. En particular se utilizan los estereotipos, que son capaces de contemplar una visión inicial de los procesos de negocio, siendo posible capturar de forma significativa eventos, entradas, recursos y salidas asociadas a un proceso de negocio. En [Dapena02] presenta una definición del modelo del negocio y del dominio utilizando Razonamiento Basado en Casos (RBC), y propone obtener el Modelo del Negocio a partir de un conjunto de especificaciones iniciales brindadas por los analistas. En [Sinogas01] proponen una extensión a UML a través de un Profile para el modelo de negocio donde específicamente definen los procesos de negocio, su relación con los objetivos y los recursos.

1.3.3. Publicaciones

- Marcela Daniele, Paola Martellotto, Gabriel Baum. Capítulo Nº XII: “Generic Model of the Business Model and its Formalization in Object-Z”, en el libro “Verification, Validation and Testing in Software Engineering”. Editores Aristides Dasso and Ana Funes. Editorial Idea Group Inc. Copyright 2007. ISBN: Hard cover: 1-59140-851-2. Soft cover:1-59140-852-0. E-book: 1-59140-853-9.

- Marcela Daniele, Paola Martellotto. “Un profile UML para el Modelo de Negocio”. Publicado en los anales del Workshop de Ingeniería de Software y Base de Datos, CACIC 2006, realizado en la Universidad Nacional de San Luis del 17 al 21 de octubre de 2006.
- Marcela Daniele, Paola Martellotto, Gabriel Baum. “Formalización del Modelo Genérico del Modelo de Negocio”, publicado en los anales de la 34 JAIIO (Jornadas Argentinas de Informática e Investigación Operativa), ISSN 1666 1141. ASIS 2005, Argentine Symposium on Information Systems, Rosario, Argentina. September 29-30, 2005.
- Marcela Daniele, Paola Martellotto, Gabriel Baum. “Traducción del Modelo Genérico del Modelo de Negocio a Objetc-Z”. Publicado en los anales del VII Workshop de Investigadores en Ciencias de la Computación (WICC 2005), Universidad Nacional de Río Cuarto, Argentina, Mayo 2005. ISBN: 950-665-337-2.
- Gabriel Baum, Marcela Daniele, Paola Martellotto, María Marta Novaira. “Un modelo Genérico para el Modelo de Negocio”. Publicado en los anales del Workshop de Ingeniería de Software y Base de Datos, CACIC 2004, realizado en la Universidad Nacional de La Matanza del 4 al 8 de octubre de 2004. [BDMN04b]
- Gabriel Baum, Marcela Daniele, Paola Martellotto, María Marta Novaira. “Un modelo Genérico para el Modelo de Negocio”. Publicado en los anales del Workshop de Investigadores en Ciencias de la Computación, (WICC 2004), realizado en la Universidad Nacional del Comahue en junio/2004. [BDMN04a]
- Gabriel Baum, Marcela Daniele, Nazareno Aguirre, Lucio Guzmán. Poster publicado en los resúmenes del Seminario Académico-Científico 2003, “Trabajos Finales de Grado, Tesis de Posgrado, Proyectos de Investigación y Trabajos de Becarios de Investigación”, organizado por la Secretaría de Ciencia y Técnica, Escuela de Posgraduación, Secretaría Académica y Secretaría de Extensión y Desarrollo de la UNRC. 20 y 21 de agosto de 2003.

1.3.4. Organización de la tesis

Los capítulos restantes de esta tesis están organizados de la siguiente forma: en el capítulo 2 se introducen los conceptos básicos del lenguaje de modelado UML, y en particular, se describe el diagrama de clases y sus características. Además se describe el lenguaje formal Object-Z, y se realiza un breve análisis de la integración entre ambos lenguajes. En el capítulo 3 se presenta una descripción resumida del Proceso Unificado, sus características, fases y flujos de trabajo. En particular, se introduce la definición de modelo de negocio y sus artefactos de acuerdo a la propuesta original del RUP. Una descripción más detallada del Proceso Unificado

puede encontrarse en el anexo I de esta tesis. En el capítulo 4 se presenta nuestra propuesta del Modelo Genérico para el modelo de negocio, mostrado con un diagrama de clases y reglas adicionales definidas en lenguaje natural. Además se desarrolla un caso de estudio que resulta de la aplicación del modelo genérico a un caso particular. En el capítulo 5 se expone la formalización del modelo genérico, usando el lenguaje de especificación formal Object-Z. En el capítulo 6 se exponen conclusiones y trabajos futuros.

Por último se lista la bibliografía referenciada y consultada.

CAPITULO 2: El Arte de Modelar

2.1. ¿ Qué es un modelo?

El proceso de desarrollo de software comienza con la construcción de un modelo, el cual representa una especificación precisa de las necesidades del usuario trasladadas a los requerimientos que el sistema debe satisfacer.

Un modelo es una simplificación de la realidad. Proporciona los planos de un sistema, detallados o más generales, ofreciendo una visión global del mismo. Todo sistema puede ser descrito desde diferentes perspectivas utilizando modelos. Un modelo puede ser estructural, destacando la organización del sistema, o de comportamiento, resaltando su dinámica.

Los modelos son abstracciones que incluyen lo esencial de un problema complejo de manera que sea más fácil de entender. Para el desarrollo de sistemas de software se deben abstraer diferentes vistas del sistema, construir modelos utilizando notaciones no ambiguas, verificar que los modelos satisfacen los requisitos del sistema, y añadir de forma gradual detalles que transforman el modelo en una implementación.

Es natural asumir que a mayor complejidad del software a desarrollar mayor será la necesidad de contar con buenas técnicas de modelado, con el fin de mejorar la comunicación entre los miembros del equipo de trabajo y asegurar la solidez de la arquitectura del sistema en desarrollo. Del análisis de requisitos surge un conjunto de modelos que ofrecen descripciones abstractas del sistema a desarrollar.

2.1.1. Propósitos del Modelado

Los modelos permiten una mejor comprensión del sistema en desarrollo. En general, podemos decir que los modelos nos ayudan a organizar, visualizar, comprender y crear los sistemas. Los propósitos básicos del modelado son:

- Ayudar a visualizar un sistema desde varias perspectivas haciendo más sencillo su desarrollo. La especificación del producto permite definir las necesidades de los usuarios.
- Especificar la estructura y el comportamiento del sistema y proporcionar plantillas que guíen la construcción del sistema. Forman un poderoso medio de comunicación y negociación entre usuarios y desarrolladores, y entre distintos desarrolladores entre sí.

- Documentar decisiones adoptadas durante el desarrollo del sistema. Permite introducir modificaciones en el sistema, ya sea para la corrección de errores o para adaptarlo a cambios en los requerimientos, dejando una prueba de que la nueva implementación corrige los errores contenidos en la versión anterior del sistema, o también para reflejar que el sistema se adapta consistentemente a los cambios de requerimientos.

2.1.2. Perspectivas de Modelado

En el modelado de software, existen básicamente dos formas de enfocar un modelo de software, la perspectiva orientada a objetos y la perspectiva algorítmica.

Hasta los años 70 la estrategia de diseño de software más utilizada se basaba en descomponer el diseño en partes funcionales con la información del sistema almacenada en un área compartida de datos (diseño funcional).

La visión tradicional del desarrollo de software toma una perspectiva algorítmica. En este enfoque, el bloque principal de construcción del software es el procedimiento o función. Esta visión conduce a los desarrolladores a centrarse en cuestiones de control y de descomposición de algoritmos grandes en otros más pequeños. Se ha comprobado que con este enfoque a medida que los sistemas crecen y cambian, su mantenimiento resulta muy complejo.

La visión del desarrollo de software fue cambiando hasta tomar una perspectiva orientada a objetos. En este enfoque, el principal bloque de construcción de todos los sistemas de software es el objeto o clase. Un objeto es una “cosa”, generalmente extraída del vocabulario del espacio del problema o del espacio de la solución; una clase es una abstracción de un conjunto de objetos similares. Todo objeto tiene identidad (puede nombrarse o distinguirse de otros objetos), estado (generalmente hay datos asociados a él), y comportamiento (el objeto posee responsabilidades y colabora con otros objetos para ejecutar algún comportamiento específico).

Actualmente la orientación a objetos es el enfoque preponderante ya que se ha mostrado válido en la construcción de sistemas en toda clase de dominios, abarcando todo el abanico de tamaños y complejidades. Mas aún, la mayoría de los lenguajes actuales, sistemas operativos y herramientas son orientados a objetos de alguna manera, lo que todavía ofrece más motivos para pensar en términos de objetos. Hoy en día el desarrollo orientado a objetos proporciona la base fundamental para ensamblar sistemas a partir de componentes.

Para construir un *modelo* es importante elegir un lenguaje para su representación. Visualizar, especificar, construir y documentar sistemas orientados a objetos es

exactamente el propósito de UML (Unified Modeling Language) y se describe a continuación.

2.2. Notación Gráfica: el Lenguaje de Modelado Unificado.

El Lenguaje de Modelado Unificado [BRJ99] es un lenguaje visual para especificar, construir y documentar los artefactos de un sistema. Es una notación de propósito general que puede ser utilizado en diferentes dominios de aplicación (ej. financiero, comercial, telefonía) y en diferentes plataformas (Ej. J2EE, .NET).

La notación UML deriva y unifica las tres metodologías de análisis y diseño orientadas a objetos más extendidas: la metodología de Grady Booch para la descripción de conjuntos de objetos y sus relaciones, la técnica de modelado orientada a objetos de James Rumbaugh (OMT: Object-Modeling Technique), y el método de casos de uso de Ivar Jacobson (OOSE: Object- Oriented Software Engineering). El desarrollo de UML comenzó a finales de 1994 cuando Grady Booch y James Rumbaugh de Rational Software Corporation empezaron a unificar sus métodos. A finales de 1995, Ivar Jacobson y su compañía Objectory se incorporaron a Rational, aportando el método OOSE. En noviembre de 1997, la OMG[1] (Object Management Group) aprueba la especificación de UML 1.1. como notación estándar para el modelado del análisis y diseño orientado a objetos. Las sucesivas revisiones y modificaciones realizadas a esta versión de UML fueron generando nuevas especificaciones [UML1]. En septiembre de 2003 la OMG da a conocer la especificación completa para la nueva versión UML 2.0. [UML2].

UML proporciona una manera estándar para escribir los planos de un sistema, cubriendo tanto los conceptuales, tales como procesos del negocio y funciones del sistema, como los concretos, tales como las clases escritas en un lenguaje de programación específico, esquemas de bases de datos y componentes de software reutilizables. Es un lenguaje muy expresivo, que cubre todas las vistas necesarias para desarrollar sistemas. Además, UML es fácilmente adaptable a nuevas técnicas y dominios ya que dispone de mecanismos de extensión que no necesitan redefinir el núcleo UML.

UML define su meta-modelo en su propia notación. Este meta-modelo es auto-referencial, es decir que cualquier lenguaje de modelado de propósito general debería

[1] El OMG (Object Management Group) se formó en 1989 con el propósito de crear una arquitectura estándar para objetos distribuidos en redes (componentes). En Octubre de 1989 empezó a funcionar como una corporación independiente y sin ánimo de lucro. El compromiso asumido por el OMG es buscar el desarrollo de especificaciones para la industria del *software* que sean técnicamente “excelentes", comercialmente viables e independientes del vendedor.

ser capaz de modelarse a sí mismo. La definición de meta-modelo se ubica en una arquitectura de cuatro capas, como se muestra en la siguiente tabla:

Capa	Descripción	Ejemplo
Meta-meta-modelo	Define el lenguaje para especificar meta-modelos. La especificación dada por el MOF (Meta Objet Facility) de OMG.	MetaClass, MetaAttribute, MetaOperation
Meta-modelo	Define el lenguaje para especificar modelos. Es una instancia del meta-meta-modelo. (UML)	Class, Attribute, Operation
Modelo	Define el lenguaje para describir un dominio de información. Es una instancia del meta-modelo.	Cliente, Nombre, DNI GetAllName()
Objetos (datos)	Define los datos específicos del usuario. Es una instancia del Modelo.	c1, José Danez, 12.221.345

Retornando a la definición de UML dada en el primer párrafo de esta sección, se explica brevemente a continuación:

- **UML es un lenguaje**

Un lenguaje de modelado es una notación cuyo vocabulario y reglas se centran en la representación conceptual y física de un sistema. UML proporciona un lenguaje de modelado estándar para visualizar, especificar, construir y documentar los artefactos de un software.

- **UML es un lenguaje para visualizar**

Un modelo gráfico facilita la comunicación. La notación de UML posee una semántica bien definida que mejora la comunicación y el entendimiento común entre desarrolladores de un producto de software.

- **UML es un lenguaje para especificar**

En este contexto, especificar significa construir modelos precisos, no ambiguos y completos. UML cubre la especificación de todas los artefactos que se construyen en desarrollo de un software.

- **UML es un lenguaje para construir**

UML no es un lenguaje de programación visual, pero sus modelos puede usarse de manera directa para obtener el código en una gran variedad de lenguajes de programación, esto significa que es posible establecer correspondencias desde un

modelo UML a un lenguaje de programación, esto se denomina ingeniería directa. La ingeniería inversa también es posible, se puede reconstruir un modelo UML a partir de una implementación orientada a objetos. En ambos casos será deseable disponer de herramientas automáticas que lo soporten y la intervención humana.

- **UML es un lenguaje para documentar**

UML cubre la documentación de la arquitectura de un sistema y todos sus detalles, proporciona un lenguaje formal para expresar en varios casos requisitos y pruebas, y permite modelar y documentar las actividades de planificación de proyectos y gestión de versiones.

2.3. Un modelo conceptual de UML

Para una mejor comprensión de UML, se requiere de un modelo conceptual del lenguaje que incluye los tres elementos principales que lo componen: los bloques básicos de construcción de UML (figura 2), las reglas que dictan cómo se pueden combinar estos bloques básicos y algunos mecanismos comunes que se aplican a través de UML.

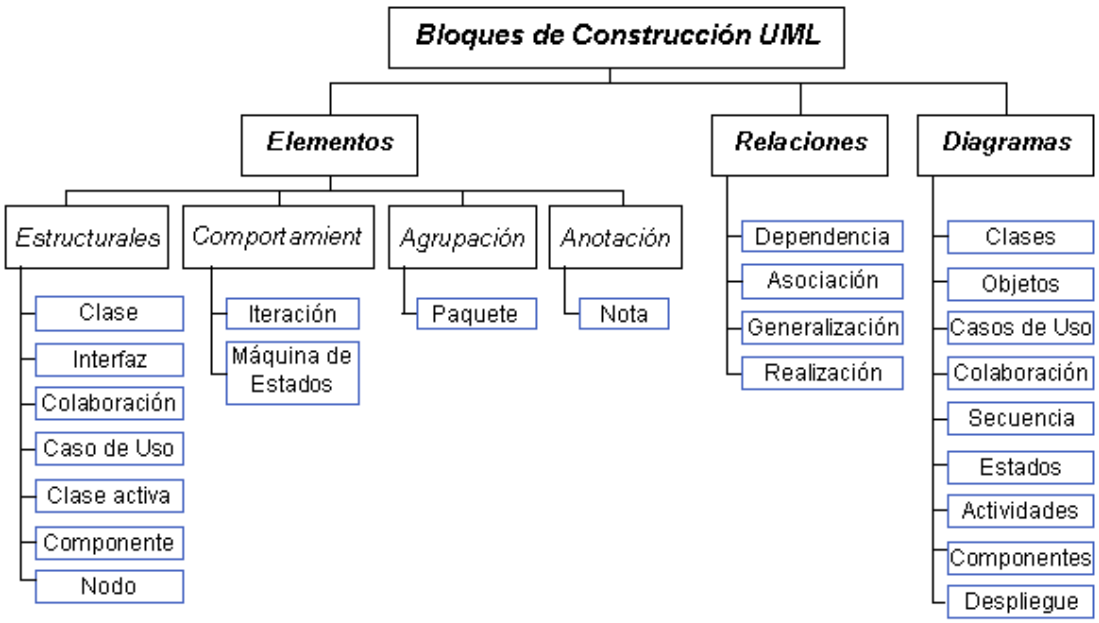


Figura 2: Bloques básicos de Construcción de UML

A continuación se describen cada uno de los componentes que conforman los bloques básicos de Construcción de UML:

2.3.1. Elementos

- **Elementos Estructurales**

Son los elementos que permiten identificar los modelos UML. Incluyen:

- Clase: descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica.
- Interfaz: signatura de una colección de operaciones que especifican un servicio de una clase o componente en forma parcial o completa. Define las especificaciones de las operaciones, no sus implementaciones.
- Colaboración: define una interacción y es una sociedad de roles y otros elementos que colaboran para proporcionar un comportamiento cooperativo mayor que la suma de los comportamientos de sus elementos. Tiene una dimensión tanto estructural como de comportamiento.
- Caso de uso: descripción de un conjunto de secuencias de acciones que un sistema ejecuta y que produce un resultado observable de interés para un actor particular.
- Clase activa: es una clase cuyos objetos tienen uno o más procesos o hilos de ejecución y por lo tanto pueden dar origen a actividades de control. Los objetos de una clase activa representan elementos cuyo comportamiento es concurrente con otros elementos.
- Componente: es una parte física y reemplazable de un sistema que conforma un conjunto de interfaces y proporciona la implementación de dicho conjunto.
- Nodo: elemento físico que existe en tiempo de ejecución y representa un recurso computacional.

- **Elementos de Comportamiento**

Son los elementos representantes de las partes dinámicas de los modelos UML. Representan comportamiento en el tiempo y el espacio. Hay dos tipos principales de elementos de comportamiento:

- Interacción: es un comportamiento que comprende un conjunto de mensajes intercambiados entre un conjunto de objetos, dentro de un contexto particular, para alcanzar un propósito específico.
- Máquina de estados: es una especificación de un comportamiento que especifica las secuencias de estados por las que pasa un objeto durante su vida en respuesta a eventos.

- **Elementos de Agrupación**

Son los elementos que representan las partes organizativas de los modelos UML.

- Paquete: es un mecanismo de propósito general para organizar elementos en grupos. A diferencia de los componentes, que existen en tiempo de ejecución, un paquete es puramente conceptual ya que solo existe en tiempo de desarrollo.

- **Elementos de Anotación**

Son partes explicativas de los modelos UML, son restricciones y comentarios que se pueden aplicar para describir, clarificar y hacer observaciones sobre los elementos de un modelo. El tipo principal de elemento de anotación es llamado Nota.

2.3.2. Relaciones

Hay cuatro tipos de relaciones:

- Dependencia: es una relación semántica entre dos elementos en la cual un cambio a un elemento (el elemento independiente) puede afectar la semántica del otro elemento (el elemento dependiente).
- Asociación: es una relación estructural que describe un conjunto de enlaces; un enlace es una conexión entre objetos.
- Generalización: es una relación de especialización/generalización en la cual los objetos del elemento especializado (el hijo) pueden sustituir a los objetos del elemento generalizado (el padre).
- Realización: es una relación semántica entre clasificadores, donde un clasificador2 especifica un contrato que otro clasificador garantiza que cumplirá.

2.3.3. Diagramas

Un diagrama es la representación gráfica de un conjunto de elementos, mayormente representado como un grafo conexo de nodos (elementos) y arcos (relaciones). Un diagrama muestra una proyección de un sistema, y representa una vista simplificada de la realidad, que puede ser: estructural, destacando la organización del sistema, o de comportamiento, resaltando la dinámica del sistema.

Los principales diagramas que incluye UML se describen brevemente a continuación y se agrupan según si modelan estructura o comportamiento del sistema.

[2] Un clasificador es un mecanismo de UML que describe características estructurales y de comportamiento. Los clasificadores incluyen clases, interfaces, tipos de datos, señales, componentes, nodos, casos de uso y subsistemas.

Diagramas que permiten modelar la estructura de un sistema:

- Diagrama de clases: muestra un conjunto de clases, interfaces, colaboraciones y sus relaciones.
- Diagrama de objetos: muestra un conjunto de objetos y sus relaciones.
- Diagrama de componentes: muestra un conjunto de componentes y sus relaciones.
- Diagrama de despliegue: muestra un conjunto de nodos y sus relaciones.

Diagramas que permiten modelar el comportamiento de un sistema:

- Diagrama de casos de uso: muestra un conjunto de casos de uso, actores y sus relaciones.
- Diagrama de secuencia: muestra una iteración, destacando la organización temporal de los mensajes enviados entre los objetos.
- Diagrama de colaboración: muestra una interacción, destacando la organización estructural de los objetos que envían y reciben mensajes.
- Diagrama de estados: muestra una máquina de estados, destacando el comportamiento dirigido por eventos de un objeto.
- Diagrama de actividades: muestra una máquina de estados, destacando el flujo de control a través de las actividades.

2.4. Reglas de UML

Los bloques construcción de UML no pueden combinarse de cualquier manera. Como cualquier lenguaje, UML tiene un número de reglas que especifican un modelo bien formado. Las reglas semánticas que proporciona UML son:

- Nombre: cómo llamar a los elementos, relaciones y diagramas.
- Alcance: el contexto que da un significado específico a un nombre.
- Visibilidad: cómo se pueden ver y utilizar esos nombres por otros.
- Integridad: cómo se relacionan consistentemente unos elementos con otros.
- Ejecución: qué significa ejecutar o simular un modelo dinámico.

2.5. Mecanismos comunes en UML

Existen cuatro mecanismos comunes que se aplican de forma consistente a través de todo el lenguaje: especificaciones, adornos, divisiones comunes y mecanismos de extensibilidad.

Las especificaciones indican que detrás de la notación gráfica de cada elemento hay una especificación que proporciona una explicación textual de la sintaxis y semántica de ese bloque de construcción.

Los adornos, pueden añadirse a los símbolos básicos que representan los elementos en la notación UML. Por ejemplo, el símbolo que indica la visibilidad de atributos u operaciones es un adorno.

Las divisiones comunes de los modelos de sistemas orientados a objetos pueden ser, al menos, de un par de formas:

- Clases y objetos: una clase es una abstracción; un objeto es una manifestación concreta de esa abstracción.
- Interfaz e implementación: una interfaz declara un contrato, y una implementación representa una realización concreta de ese contrato.

Los mecanismos de extensibilidad representan un medio poderoso de UML que permiten extender de manera controlada la sintaxis y la semántica del lenguaje. Los mecanismos de extensibilidad de UML son tres: *estereotipos*, que extienden el vocabulario de UML permitiendo crear nuevos tipos de bloques de construcción que derivan de los existentes, pero son específicos para un problema, representan nuevos elementos de modelado; *valores etiquetados*, que extienden las propiedades de un bloque de construcción de UML permitiendo añadir nueva información a la especificación del elemento, representan nuevos atributos de modelado; y las *restricciones*, que extienden la semántica de un bloque de construcción de UML permitiendo añadir nuevas reglas o modificar las existentes, representan nueva semántica de modelado.

A continuación se describen con mayor amplitud los diagramas de clases de UML por ser estos los utilizados en el capítulo 4 de este trabajo. Para mayor detalle de UML el elector puede acudir al libro [BRJ99] o a la especificación formal del lenguaje dada por la OMG [OMG].

2.6. Diagrama de Clases

2.6.1. Introducción

Los diagramas de clases permiten modelar la vista estática y lógica de un sistema. Son principalmente usados para modelar el vocabulario del sistema, para modelar colaboraciones simples y para modelar esquemas lógicos de bases de datos. Estos diagramas, son importantes tanto para visualizar, especificar y documentar modelos estructurales, como para construir sistemas ejecutables a través de ingeniería directa e inversa.

Un diagrama de clases está formado por un conjunto de clases, interfaces, colaboraciones y relaciones, y gráficamente consiste en un grafo orientado. A continuación se detallan los posibles usos de un diagrama de clases y los elementos que lo componen.

2.6.2. Usos de un Diagrama de Clases

Modelar el vocabulario del sistema: involucra tomar decisiones sobre qué abstracciones son parte del sistema que se está modelando y cuáles caen fuera de sus límites. Estos diagramas se usan para especificar estas abstracciones y sus responsabilidades.

Las clases se usan muy comúnmente para modelar abstracciones encontradas en el problema que se esta tratando de resolver, o en la tecnología que se esta usando para implementar la solución. Cada una de estas abstracciones es una parte del vocabulario del sistema, y juntas representan los elementos que son importantes para los usuarios y los desarrolladores.

Para los usuarios la mayoría de las abstracciones no son difíciles de identificar dado que típicamente, representan cosas que ellos ya usan para describir sus sistemas. Para los desarrolladores estas abstracciones son generalmente elementos de la tecnología que son parte de la solución.

Para modelar el vocabulario de un sistema se deben:

- Identificar aquellas los elementos usados por usuarios y/o desarrolladores para describir el problema y la solución.
- Identificar un conjunto de responsabilidades, para cada abstracción. Asegurar que cada clase está bien definida y que las responsabilidades están bien balanceadas entre ellas.
- Proveer a los atributos de las operaciones necesarias para llevar a cabo las responsabilidades de cada clase.

Modelar colaboraciones simples: una colaboración es una sociedad de clases, interfaces y otros elementos que trabajan juntos para proveer algún comportamiento cooperativo. Los diagramas de clases se usan para visualizar estos conjuntos de clases y sus relaciones.

Ninguna clase aislada tiene valor, cada una debe trabajar en colaboración con otras. Por ello, además de capturar el vocabulario de un sistema se deberá prestar atención a la visualización, especificación, construcción y documentación de las distintas formas en que los elementos interactúan. Cuando se crea un diagrama de clases, solo se modela una parte de los elementos y relaciones que componen una vista de diseño del sistema, por lo tanto deben enfocarse en una colaboración cada vez.

Para modelar una colaboración se deben:

- Identificar el mecanismo que se quiere modelar. Un mecanismo representa una función o comportamiento de la parte del sistema que se esta modelando y que es el resultado de la interacción de una sociedad de clases, interfaces y otros elementos.
- Identificar las clases, interfaces y otras colaboraciones que participan en cada colaboración, e identificar las relaciones entre estos elementos.
- Usar escenarios para recorrer la interacción entre los elementos. De esta forma, se pueden descubrir partes faltantes o erróneas en el modelo.
- Asegurar una asignación balanceada de responsabilidades entre clases, y más tarde convertir dichas responsabilidades en atributos y operaciones concretas.

Modelar el esquema lógico de una base de datos: en muchos dominios, es necesario almacenar información persistente en una base de datos para ser recuperada posteriormente. Los diagramas de clases pueden ser usados para especificar el esquema lógico de la base de datos, así como base de datos físicas.

Para modelar un esquema de bases de datos se deben:

- Identificar aquellas clases del modelo cuyo estado debe trascender el tiempo de vida de las aplicaciones.
- Crear un diagrama de clases que contenga estas clases y marcarlas como persistentes.
- Especificar los detalles de los atributos de las clases, sus asociaciones y cardinalidad.

- Crear abstracciones intermedias para simplificar la estructura lógica de la base datos, cuando existan patrones comunes que podrían complicar el diseño de la base de datos física, tales como asociaciones cíclicas y n-arias.
- Considerar el comportamiento de las clases, agregando operaciones que sean útiles para el acceso y la integridad de los datos.

2.6.3. Elementos de un Diagrama de Clases

2.6.3.1. Clase

Una clase es un tipo particular de clasificador, considerado el bloque básico de construcción de cualquier sistema orientado a objetos. Es una descripción de un conjunto de objetos, que comparten los mismos atributos, operaciones y semántica. Las clases se utilizan para identificar el vocabulario del sistema que se esta modelando, pueden incluir abstracciones como parte del dominio del problema, o parte de una implementación. Se pueden usar clases para representar elementos de hardware, software e incluso elementos que son puramente conceptuales. Las clases bien estructuradas tienen limites bien definidos y forman parte de una distribución de responsabilidades bien balanceadas a lo largo del sistema.

Gráficamente, una clase es representada por un rectángulo que posee tres divisiones o compartimentos, como se muestra en la figura 3.

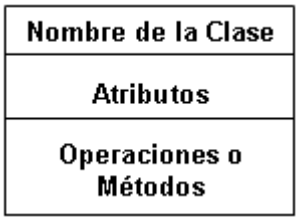


Figura 3: Elementos de una clase

Nombre de la clase. Toda clase debe tener un nombre que la distinga de las otras clases. El nombre debe ser significativo, debe dar idea del concepto que la clase representa.

Atributos. Variables de instancia que caracterizan a la Clase. Un atributo es una propiedad con nombre y describe un rango de valores que las instancias de la propiedad pueden asumir. Un atributo es una abstracción del tipo de dato o estado en que un objeto de la clase puede encontrarse. En un momento en el tiempo, un objeto de una clase tendrá valores específicos para cada uno de sus atributos. Una clase puede tener cualquier número de atributos.

Métodos u operaciones. Representan la forma en que interactúa el objeto con su entorno (dependiendo de la visibilidad: private, protected o public). Una operación es la

implementación de un servicio que puede ser requerido a cualquier instancia de la clase para realizar un comportamiento. Una clase puede tener cualquier número de operaciones. A menudo, la invocación de una operación en un objeto implica el cambio de los datos o del estado del objeto.

2.6.3.2. Relaciones

Cuando se construyen abstracciones, muy pocas veces las clases están aisladas, generalmente colaboran unas con otras en diversas formas. Cuando se modela un sistema se identifican los elementos que forman el vocabulario del sistema y como estos se relacionan entre sí.

Los tipos básicos de relaciones entre clases en el modelado orientado a objetos fueron introducidos en la sección 2.3.2, se describen con mayor detalle a continuación:

- **Dependencia**

Una dependencia es una relación de uso, una relación semántica entre dos elementos, en la cual un cambio en un elemento, considerado independiente, puede afectar la semántica del otro elemento, considerado dependiente.

En UML, una dependencia se representa gráficamente como una línea discontinua dirigida desde la clase independiente a la dependiente. (figura 4).

En algunos casos es necesario especificar ciertos matices de una relación de dependencia, para lo cual se le agrega algún estereotipo. UML provee más de 15 estereotipos (send, bind, derive, extend, include, call, trace, etc) que pueden ser aplicados a las dependencias y se especifica gráficamente sobre la línea que indica la dependencia.

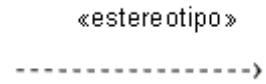


Figura 4: Relación de Dependencia

- **Generalización**

La generalización es un tipo de relación dada entre una clase general, llamada superclase o clase padre, y un tipo más especifico de esa clase, llamada subclase o clase hijo. Es usada para expresar que las subclases son casos particulares de la superclase. Expresan la relación “es_un” entre un hijo y un padre. La subclase hereda todas las propiedades de su superclase. A menudo, al hijo se le adicionan atributos y operaciones propias.

En UML, una relación de generalización puede contener algunos adornos, como alguno de los estereotipos implementation, complete, incomplete, disjoint, overlapping.

Gráficamente, la relación de generalización se denota por una flecha de línea sólida con punta cerrada dirigida hacia el padre. (Figura 5).

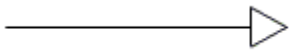


Figura 5: Relación de Generalización

- **Asociación**

Una asociación es una relación estructural entre objetos de la misma o de distintas clases. Una asociación entre dos clases A y B indica que es posible navegar desde una instancia de A hacia una instancia de B, y viceversa. Es posible, también, que ambos extremos de la asociación correspondan a la misma clase. Esto se conoce como asociación recursiva y significa que un objeto de una clase dada puede conectarse con otros objetos de la misma clase.

Gráficamente, la asociación se denota por una línea sólida que conecta una clase a otra. (Figura 6).

Además, una asociación puede tener otros adornos, tales como nombre, dirección del nombre, roles, multiplicidad y navegabilidad. El nombre es usado para describir la naturaleza de la relación y la dirección del nombre especifica en que sentido se debe leer dicho nombre y se indica con una punta de flecha rellena. El rol se indica en un final de asociación y define el papel que juega la clase respecto de la clase que se encuentra en otro extremo de la relación. La multiplicidad de una asociación determina cuantos objetos pueden estar conectados con otros en una instancia de asociación. La multiplicidad se puede denotar como expresiones que toman valor en un rango o como valores explícitos. Cuando se agrega multiplicidad a un final de asociación, esta especificando cuantos objetos del tipo de este final de asociación puede tener asociado cada objeto del otro final de asociación. La navegación se denota con una punta de flecha en un extremo de la asociación e indica que se podrá navegar hacia los objetos de la clase apuntada pero no a la inversa. Si no hay flechas en la asociación, entonces la navegación es bidireccional.

Figura 6: Relación de Asociación

- **Agregación**

En algunas situaciones es necesario modelar relaciones del tipo “todo/parte”, en la cuales una clase representa un elemento, el “todo”, que consiste de elementos mas pequeños, las “partes”. Una agregación representa una relación “tiene un”. La agregación es un tipo especial de asociación y se denota adornando a la asociación con un diamante sin relleno al final de asociación de la clase que representa el todo, llamado agregación simple. La agregación simple es completamente conceptual y solo distingue un “todo” de sus “partes”.

La composición o agregación compuesta, es una forma de agregación con una fuerte relación de pertenencia y vidas coincidentes de cada “parte” con el “todo”. Las “partes”, con una multiplicidad no fijada, pueden crearse después que el “todo” al que pertenecen, pero una vez creadas viven y mueren con él. Las “partes” pueden eliminarse explícitamente antes de eliminar la parte compuesta. Gráficamente, se denota adornando a la asociación con un diamante relleno. Claro está que si la agregación es un tipo de asociación, puede contener todas las propiedades enunciadas de una asociación.

- **Realización**

Es una relación semántica entre clasificadores, donde un clasificador especifica un contrato que otro clasificador garantiza que cumplirá. Gráficamente se denota con una línea discontinua y una gran punta de flecha cerrada indicando la clase que realiza (Figura 7).

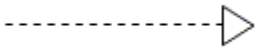


Figura 7: Relación de Realización

- **Clase Asociación**

Es una asociación entre dos clases que tiene propiedades de clase. Gráficamente se denota como una clase que se relaciona a través de una línea punteada a la asociación entre las clases que la originan. (Figura 8).

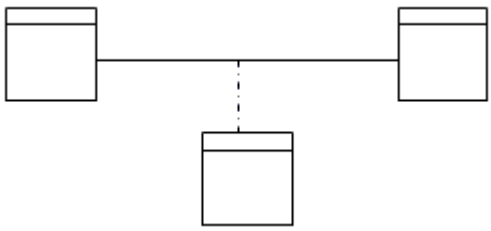


Figura 8: Clase Asociación

2.6.3.3. Interfaz

Es una colección de operaciones que se usa para especificar un servicio de una clase o componente. Tiene un nombre, cadena de cualquier longitud con letras y números, y generalmente está precedida de la letra I, para diferenciarla de las clases.

En UML, se utilizan las interfaces para modelar las líneas entre la especificación de lo que una abstracción hace y la implementación de cómo lo hace.

Generalmente, se conecta a la clase o componente que la implementa a través de una relación de realización, y a la clase o componente que utiliza sus servicios se conecta con una dependencia. Gráficamente, una interfaz puede representarse de forma icónica o en forma expandida como una clase estereotipada (Figura 9).

Las interfaces que una clase o componente realiza se llaman de exportación y en las que se basa se llaman de importación.

Figura 9: Interfaz en forma icónica y en forma expandida

2.7. UML Profile[3]

Como ya se mencionó, UML es un lenguaje de propósito general que proporciona una gran flexibilidad y expresividad a la hora de modelar sistemas. Sin embargo, existen situaciones donde es necesario contar con un lenguaje que modele y represente de forma específica un cierto dominio particular. De acuerdo a lo propuesto por la OMG, una posibilidad es definir un nuevo lenguaje específico para el dominio que se quiere modelar, mientras que la otra es extender UML a través de la definición de un profile.

Un UML profile provee un mecanismo de extensión genérico para construir modelos UML en un dominio particular. Se basa en un conjunto de estereotipos y valores etiquetados adicionales que se aplican a elementos, atributos, métodos y vínculos, con la intención de extender el lenguaje UML para adaptar su uso en un dominio, plataforma o método particular.

En la especificación de la infraestructura de UML 2.0 [UML2] se resumen las diferentes razones por las que un desarrollador querría adaptar un metamodelo, y se transcriben brevemente a continuación:

[3] No se traduce la palabra profile en todo el texto de este trabajo.

- Disponer de una terminología propia para un dominio o plataforma particular.
- Definir una sintaxis para construcciones que no cuentan con una notación propia (como sucede con las acciones).
- Definir una notación diferente para símbolos ya existentes.
- Agregar semántica que no está determinada de manera precisa en el metamodelo o que no existe en el metamodelo.
- Añadir restricciones que limiten la forma de usar el metamodelo y sus constructores.
- Añadir información que puede ser útil cuando se transforma un modelo en otros modelos, o a código.

En la definición de la arquitectura dada por la OMG y expuesta en este trabajo en la sección 2.2., se distinguen los cuatro niveles de abstracción que participan en el modelado de un sistema: el nivel de las instancias, el nivel del modelo del sistema, el nivel del modelo del modelo del sistema o metamodelo y el nivel del meta-metamodelo. Conocidos comúnmente como nivel M0, M1, M2 y M3, respectivamente. La definición de profiles es realizada en el nivel M2, donde se definen los elementos que intervienen a la hora de definir un modelo concreto de un sistema en el nivel M1, es decir, los elementos del nivel M1 pueden ser considerados como *instancias* de los elementos del nivel M2.

2.7.1. Definición

Un profile se define en un paquete UML, estereotipado «profile», que extiende a un metamodelo o a otro profile. Se utilizan los tres mecanismos de extensibilidad de UML para definir profiles: estereotipos *(stereotype*s), restricciones *(constraint*s), y valores etiquetados *(tagged value*s):

- Un estereotipo define un conjunto de propiedades que poseen sus elementos. Son usados para introducir elementos al metamodelo. Un estereotipo se define por un nombre, y por una serie de elementos del metamodelo sobre los que puede asociarse. Gráficamente, un estereotipo se representa igual que una clase más la palabra reservada «stereotype», ubicada sobre el nombre de la clase.
- Las restricciones, se asocian a los estereotipos, y son propiedades que especifican semántica o condiciones que deben ser mantenidas verdaderas para todos los elementos del metamodelo. Las restricciones pueden

expresarse tanto en lenguaje natural como en algún lenguaje formal o semiformal.

- Un valor etiquetado es un meta-atributo adicional que se asocia a una metaclase del metamodelo extendido por un profile. Todo valor etiquetado debe contar con un nombre y un tipo, y se asocia a un determinado estereotipo. Los valores etiquetados se representan de forma gráfica como atributos de la clase que define el estereotipo.

Es importante señalar que estos tres mecanismos de extensión no son de primer nivel, es decir, no permiten modificar metamodelos existentes, sólo añadirles elementos y restricciones, pero respetando su sintaxis y semántica original. Sin embargo, son muy adecuados para particularizar un metamodelo para uno o varios dominios o plataformas existentes.

Existe una gran cantidad de profiles definidos en la actualidad, algunos de ellos han sido estandarizados por la OMG [OMG], otros en proceso de serlo y otros que están puestos a disposición para ser utilizados, a pesar de no representar estándares. Algunos UML profiles son: para CORBA y para CCM (CORBA Component Model), para EDOC (Enterprise Distributed Object Computing), para EAI (Enterprise Application Integration), para Planificación, Prestaciones, y Tiempo (Scheduling, Performance, and Time), para SPEM v1.0 (beta) (from "Software Process Engineering Metamodel Specification" by the OMG), para la definición de procesos y sus componentes utilizando la especificación de SPEM, para XSD Schema ("Modeling XML Applications with UML" de David Carlson), para Web Modeling ("Building Web Applications with UML" de Jim Conallen), para trabajar con páginas web, servers, scripts, ASP, JSP, etc., para Business Process Modeling, para Modelado de Negocios derivado de la extensión de Eriksson y Penker. ("Business Modeling with UML" por Hans-Erik Eriksson and Magnus Penker), el “UML/EJB Mapping Specification”, definido por JCP (Java Community Process).

2.7.2. Construir un UML Profile

En esta sección se describen los pasos necesarios para construir un *UML profile*, según [FV04]. Los pasos a seguir son:

1. Disponer de la definición del metamodelo de la plataforma o dominio de la aplicación que se pretende modelar con un UML *profile*.
2. Definir el *profile*. Para ello:
   Dentro del paquete *«profile»* incluir un estereotipo por cada uno de los elementos del metamodelo que se desea incluir en el *profile*. Estos estereotipos

tendrán el mismo nombre que los elementos del metamodelo, estableciéndose de esta forma una relación entre el metamodelo y el *profile*. En principio, cualquier elemento necesario para definir el metamodelo puede ser etiquetado posteriormente con un estereotipo.

3. Es importante tener claro cuáles son los elementos del metamodelo de UML que se está extendiendo y sobre los que es posible aplicar un estereotipo. Ejemplo de tales elementos son las clases, sus asociaciones, sus atributos, las operaciones, las transiciones, los paquetes, etc. De esta forma cada estereotipo se aplicará a la metaclase de UML que se utilizó en el metamodelo del dominio para definir un concepto o una relación.
4. Definir como valores etiquetados de los elementos del *profile* los atributos que aparecen en el metamodelo. Incluir la definición de sus tipos, y sus posibles valores iniciales.
5. Definir las restricciones que forman parte del *profile*, a partir de las restricciones del dominio. Por ejemplo, las multiplicidades de las asociaciones que aparecen en el metamodelo del dominio, o las propias reglas de negocio de la aplicación deben traducirse en la definición las correspondientes restricciones.

Una vez que se ha definido el *UML profile*, la forma de utilizarlo en una aplicación concreta se representa mediante una relación de dependencia, estereotipada «apply», entre el paquete UML que contiene la aplicación, y los paquetes que definen el UML *profile*.

2.8. El lenguaje de especificación Object-Z

La notación Z [Spivey92] está basada en la teoría de conjuntos y la lógica matemática. La teoría de conjuntos usada incluye operadores estándares de conjuntos, comprensión de conjuntos, productos cartesianos y conjuntos potencia. La lógica matemática es el cálculo de predicados de primer orden. Juntos forman un lenguaje matemático que es fácil de usar y de aplicar.

Otro aspecto importante de Z es la forma en que las matemáticas pueden ser estructuradas. Objetos matemáticos y sus propiedades pueden ser reunidos en esquemas: patrones de declaración y restricciones. El lenguaje de esquemas puede ser usado para describir el estado de un sistema y las formas en el cual dicho estado puede cambiar. Además puede ser usado para describir propiedades del sistema y para razonar acerca de posibles refinamientos de un diseño.

Un rasgo característico de Z es el uso de tipos. Cada objeto en el lenguaje matemático tiene un único tipo, representado como un conjunto máximo en la especificación actual. Existen varias herramientas de chequeo de tipos que apoyan el uso práctico de Z.

Un tercer aspecto es el uso del lenguaje natural. Se utilizan las matemáticas para exponer el problema, para descubrir soluciones y para probar que el diseño elegido cumple con la especificación. Se utiliza el lenguaje natural para relacionar las matemáticas con los objetos del mundo real. Una especificación bien escrita debería ser perfectamente obvia al lector.

Un cuarto aspecto es el refinamiento. Se realiza el desarrollo de un sistema a partir de un modelo que utiliza tipos de datos matemáticos simples para identificar el comportamiento deseado. Luego se refina esta descripción construyendo otro modelo que respete las decisiones de diseño tomadas y que esté más cerca de la implementación.

Si bien Z es un poderoso lenguaje de especificación matemático, posee un inconveniente importante y es la falta de estructura modular en las especificaciones, lo cual ocasiona una mayor complejidad en especificaciones de gran tamaño.

Object-Z [Smith00] es una extensión de Z que soporta especificaciones en un estilo orientado a objetos. Una especificación orientada a objetos describe un sistema como una colección de objetos que interactúan entre ellos, y cada uno de los cuales tiene una estructura y un comportamiento prescripto. Esta descomposición facilita la claridad de especificaciones de gran tamaño.

Una especificación Z define un número de esquemas de estados y operaciones. Para inferir cuales operaciones pueden afectar a un esquema de estado particular es necesario examinar los perfiles de cada una de las operaciones. Object-Z asocia un conjunto de operaciones individuales con un esquema de estado, es decir una operación se refiere sólo al estado del objeto al que pertenece. Para esto define el concepto de clase, permitiendo declarar objetos de esa clase y extender las clases mediante herencia.

2.8.1. Definición de Clases en Object-Z

Sintácticamente, una clase es una caja con nombre y parámetros en algunos casos. La clase declarada en Object-Z puede contener una lista de visibilidades, clases heredadas, definiciones de tipos, definiciones de constantes, un esquema de estado, un esquema inicial, esquemas de operaciones y un invariante de historias (Figura 10). La [lista de visibilidades] muestra las interfaces de una clase. El [esquema de estado] contiene declaraciones de variables de estado y el predicado del estado que

representa el invariante de la clase y está implícitamente incluido en cada esquema de operación y en el esquema inicial. El [esquema inicial] puede no ser único. Los [esquemas de operación] se representan con una caja con nombre y describen los métodos definidos por la clase, es decir, las transiciones que un objeto de la clase puede experimentar.

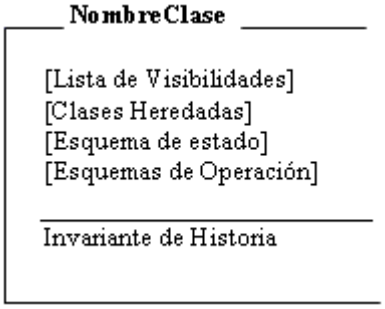


Figura 10: Componentes de una clase Object-Z

Object-Z incorpora también genericidad en sus clases, y la posibilidad de incluir invariantes de clase, es decir, predicados (invariante de historias) que se verifican a lo largo de la vida de los objetos de esa clase, y que restringen el comportamiento de los objetos. Los invariantes se expresan como fórmulas usando la lógica temporal.

La inicialización de objetos no es explícita en Object-Z, se realiza a través del esquema inicial identificado con init. Considérese la figura 11 donde se muestra un ejemplo de la especificación en Object-Z para la clase PilaGen, que define una pila genérica.

Una clase definida en Object-Z puede ser usada por instanciación o por herencia:

- Instanciación: la clase puede ser usada como un tipo de Z. Es decir, si **C** es una clase, **c:C** es una declaración del objeto **c** de la clase **C**. Entonces, si **a** es un atributo y **op** una operación ambos correspondientes a la clase **C**, entonces **c.a** denota el atributo **a** del objeto **c** que es de tipo **C**, y **c.op** es la aplicación de la operación **op** sobre el objeto **c** de tipo **C**. Para el ejemplo de la pila genérica definida por la clase PilaGen, **p1,p2 : PilaGen** representan la definición de dos objetos de la clase PilaGen, entonces p1.Poner, p1.Sacar, p2.Init y p2.Sacar son ejemplos de referencias a las operaciones de la Pila.
- Herencia: la clase puede ser reusada incorporando sus características a otras clases a través de la herencia. Por ejemplo, si se quisiera definir una pila de naturales PilaAcotada, se podrían reusar las características de la clase definida PilaGen, agregando el nombre de esta clase en la parte superior del esquema de clase que define PilaAcotada, como muestra el fragmento del esquema de

clase que la define en la figura 12, y por supuesto realizando todas las redefiniciones y agregando las características propias de la clase que se define.

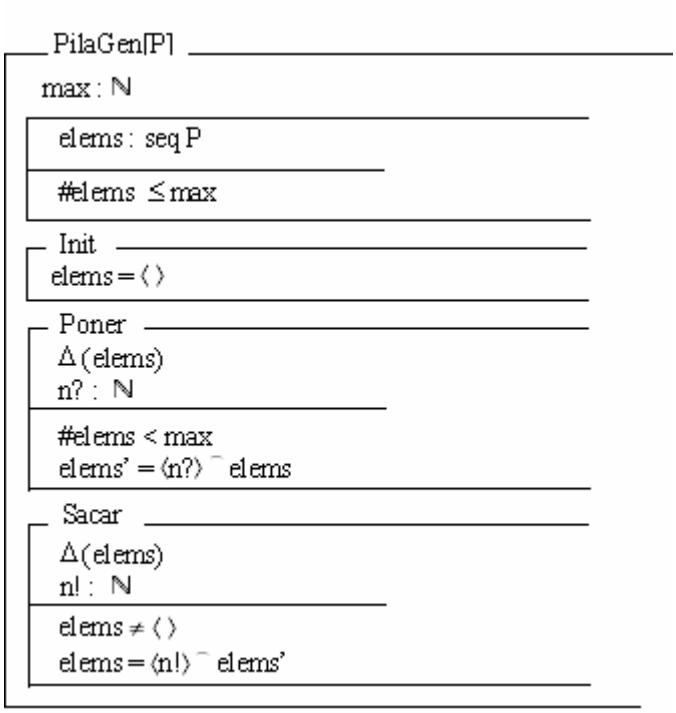


Figura 11: Esquema de Clase Object-Z de la clase PilaGen

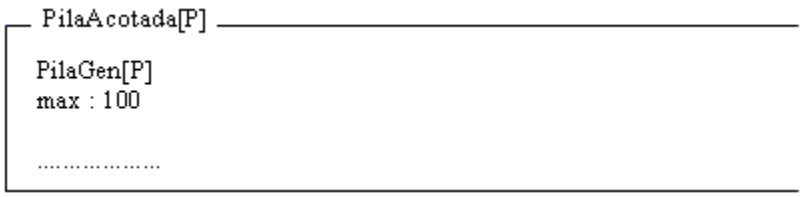


Figura 12: Fragmento del esquema de Clase Object-Z de la clase PilaAcotada que hereda de PilaGen

Cada clase declarada en Object-Z tiene implícitamente declarada una constante denominada *self* que denota el identificador de un objeto para un objeto dado.

Otros operadores de Object-Z son utilizados en este trabajo, específicamente en el capítulo 5 donde se presenta una formalización del modelo genérico desarrollado en esta tesis. En dicho capítulo se muestra una tabla con los operadores utilizados y una breve descripción de cada uno.

2.9. Modelado Gráfico (UML) vs Notación Formal (Object-Z)

UML [BRJ99] es un lenguaje de modelado gráfico que contiene un conjunto de notaciones que permiten especificar y describir un sistema desde diferentes perspectivas. Estas notaciones son fácilmente reconocidas de manera intuitiva, no

obstante UML sufre los problemas usuales de las notaciones gráficas: “carecen de una semántica precisa”. La falta de suficiente precisión en los modelos descriptos con diagramas UML puede llevar a inconsistencias entre los modelos del sistema y que diferentes desarrolladores interpretan el mismo modelo de maneras diferentes u erróneas.

En todo sistema de software aparecen restricciones que en algunos casos es difícil y en otros imposibles, expresarlos con precisión usando solo UML. El lenguaje de restricción de objetos OCL (Object Constraint Language) [OCL01], es un lenguaje textual, provisto con UML, que permite expresar restricciones o características adicionales a los modelos UML, de forma similar a la lógica de predicados. OCL es un lenguaje semi formal, dado que posee una sintaxis definida de manera precisa, pero no así su semántica que aún posee ambigüedad. La evaluación de expresiones OCL simplemente retornan un valor y no alteran el estado del sistema.

Los lenguajes de especificación formal prevén la definición de modelos completos y precisos para sistemas de software propuestos. Su objetivo es la descripción no ambigua, tanto de la estructura como de la funcionalidad de los sistemas. Como otras técnicas formales, Object-Z utiliza una estricta notación matemática y lógica que permite un riguroso análisis y razonamiento acerca de las especificaciones de los sistemas. Permite establecer una clara consistencia entre los modelos desarrollados en las diferentes etapas, principalmente entre el diseño y la implementación. Además, la especificación formal requiere de un riguroso un análisis de los requerimientos del sistema en las primeras etapas del desarrollo y la corrección de errores en etapas tempranas es muyo menos costosa que modificar el sistema cuando ya ha sido desarrollado.

La notación formal permite definir una semántica formal a la notación gráfica, y de esa manera, reducir los riesgos por imprecisiones en los modelos de sistemas de software. Además, permite aplicar técnicas de verificación y validación sobre los modelos. En todos los casos, el objetivo es desarrollar sistemas utilizando una combinación de ambas técnicas, tanto gráfica como formal, detectando cuando es conveniente usar una, otra o ambas.

En la sección 5.1. del capítulo 5 de esta tesis se describe una reseña de trabajos que proponen la definición formal de elementos del lenguaje de modelado gráfico UML al lenguaje formal Object-Z.

CAPITULO 3: El Modelado del Negocio en el Proceso Unificado

3.1. El Proceso Unificado

El Proceso Unificado [JBR99] es una metodología de desarrollo de software que define quién está haciendo qué, cuándo, y cómo para construir o mejorar un producto de software. Es una guía para todos los participantes del proyecto: clientes, usuarios, desarrolladores, directivos, ordena las actividades del equipo, dirige las tareas de cada desarrollador y del equipo como un todo. Esta metodología especifica los artefactos que deben desarrollarse en cada una de las etapas del ciclo de vida de desarrollo de un software. Además, ofrece criterios para el control y la medición, reduce riesgos y hace el proyecto más predecible.

El Proceso Unificado utiliza UML [BRJ99] como medio de expresión de los diferentes modelos que se crean durante las etapas del desarrollo.

El Proceso Unificado de Rational (Rational Unified Process - RUP) [RUP] surge como una ampliación del Proceso Objectory de Rational (Rational Objectory Process – ROP), surgido y desarrollado entre 1995 y 1997. Rational se unió con otras empresas y a partir de la unificación de diversas técnicas de desarrollo aportadas por el trabajo de muchos metodologistas, nace el llamado Proceso Unificado de Rational. Un cambio importante que distinguió el RUP del ROP fue el nuevo flujo de trabajo para el modelado de negocio.

Los tres principales aspectos que identifican y definen al Proceso Unificado se resumen en: dirigido por casos de uso, centrado en la arquitectura e iterativo e incremental.

3.1.1. El Proceso Unificado está dirigido por casos de uso

Para construir un sistema con éxito es necesario conocer lo que sus futuros usuarios necesitan y desean. El término usuario representa alguien o algo que interactúa con el sistema en desarrollo. Una interacción de este tipo es un caso de uso, definido como un fragmento de funcionalidad del sistema que proporciona al usuario un resultado importante. Los casos de uso representan los requisitos funcionales y todos juntos constituyen el modelo de casos de uso, que describe la

funcionalidad total del sistema. Basándose en el modelo de casos de uso, los desarrolladores crean los modelos de análisis, diseño e implementación que llevan a cabo los casos de uso.

3.1.2. El Proceso Unificado está centrado en la arquitectura

La arquitectura de un sistema software se describe mediante diferentes vistas del sistema en construcción. Se ve influida por muchos factores, como la plataforma (arquitectura hardware, sistema operativo, sistema de gestión de base de datos, protocolos para comunicaciones en red), consideraciones de implantación, requisitos no funcionales (rendimiento, fiabilidad). La arquitectura resalta las características más importantes de cada vista del sistema y deja de lado ciertos detalles.

La definición de la arquitectura en el desarrollo de un sistema software es importante porque facilita la comprensión del sistema por todos los que intervengan, permite organizar el desarrollo dividiendo el sistema en subsistemas, con interfaces bien definidas y una clara asignación de responsables a cada subsistema lo cual reduce la necesidad de comunicación entre los diferentes grupos de trabajo. Además, la definición de la arquitectura contribuye a la reutilización de componentes de software y la evolución favorable del sistema con flexibilidad y tolerancia a los cambios.

3.1.3. El Proceso Unificado es iterativo e incremental

El desarrollo de un producto software comercial supone un gran esfuerzo que puede durar entre varios meses hasta posiblemente un año o más. Es práctico dividir el trabajo en partes más pequeñas o miniproyectos. Cada miniproyecto es una iteración que resulta en un incremento. Las iteraciones hacen referencia a pasos en el flujo de trabajo, y los incrementos, al crecimiento del producto. Para una mayor efectividad las iteraciones deben estar controladas, seleccionadas y ejecutarse de manera planificada.

Los desarrolladores basan la selección de lo que se implementará, en una iteración que debe considerar dos factores. En primer lugar, la iteración trata un grupo de casos de uso que juntos amplían la utilidad del producto desarrollado. En segundo lugar, la iteración identifica, gestiona y reduce los riesgos más importantes en las primeras fases de inicio y elaboración.

En cada iteración, los desarrolladores identifican y especifican los casos de uso relevantes, crean un diseño utilizando la arquitectura seleccionada como guía, implementan el diseño mediante componentes, y verifican que los componentes satisfacen los casos de uso. Si una iteración cumple con su objetivo el desarrollo

continúa con la siguiente iteración. Cuando una iteración no cumple sus objetivos, los desarrolladores deben revisar sus decisiones previas y probar con un nuevo enfoque.

Un incremento es la diferencia entre dos *líneas base* sucesivas. Se llama *línea base* al estado concreto que consiguen el conjunto de modelos obtenidos al final de una iteración.

La aplicación de un método iterativo e incremental tiene beneficios importantes en el desarrollo de un sistema software entre los que se pueden destacar:

- La reducción de riesgos críticos. Si se debe retornar o repetir una iteración, solo se pierde el esfuerzo mal empleado de la iteración y no el valor del producto entero.
- Asegurar, al final de la fase de elaboración, que no existen riesgos ocultos con costos insuficientemente examinados y posibilidades de no salirse del calendario previsto. Mediante la identificación de riesgos en fases tempranas del desarrollo, el tiempo que se insume en resolverlos se emplea al principio de la planificación, cuando existe menos presión por cumplir los plazos. En el método “tradicional”, en el cual los problemas complicados se revelan por primera vez en la prueba del sistema, el tiempo necesario para resolverlos normalmente es mayor que el tiempo que queda en la planificación, y casi siempre obliga a retrasar la entrega.
- La iteración controlada acelera el ritmo del esfuerzo de desarrollo en su totalidad debido a que los desarrolladores trabajan de manera más eficiente para obtener resultados claros a corto plazo, en lugar de tener un calendario largo, que se prolonga eternamente.
- La iteración controlada reconoce que las necesidades del usuario y sus requisitos no pueden definirse completamente al principio. Típicamente, se obtienen y refinan en iteraciones sucesivas, y esta forma de operar hace más fácil la adaptación a los requisitos cambiantes. La arquitectura proporciona la estructura sobre la cual guiar las iteraciones, mientras que los casos de uso definen los objetivos y dirigen el trabajo de cada iteración.

3.2. La vida del Proceso Unificado

El Proceso Unificado está estructurado a lo largo de dos dimensiones:

- Tiempo: división del ciclo de vida en fases e iteraciones.
- Componentes del proceso: producción de un conjunto de artefactos o modelos específicos para representar un aspecto del sistema.

La dimensión del tiempo involucra las fases de *Inicio*, *Elaboración*, *Construcción* y *Transición*. La dimensión de componentes del proceso incluye las actividades de *Modelado de Negocio*, *Captura de Requerimientos*, *Análisis*, *Diseño*, *Implementación* y *Prueba*. Cada actividad de la dimensión de componentes es aplicada en cada fase de la dimensión basada en el tiempo.

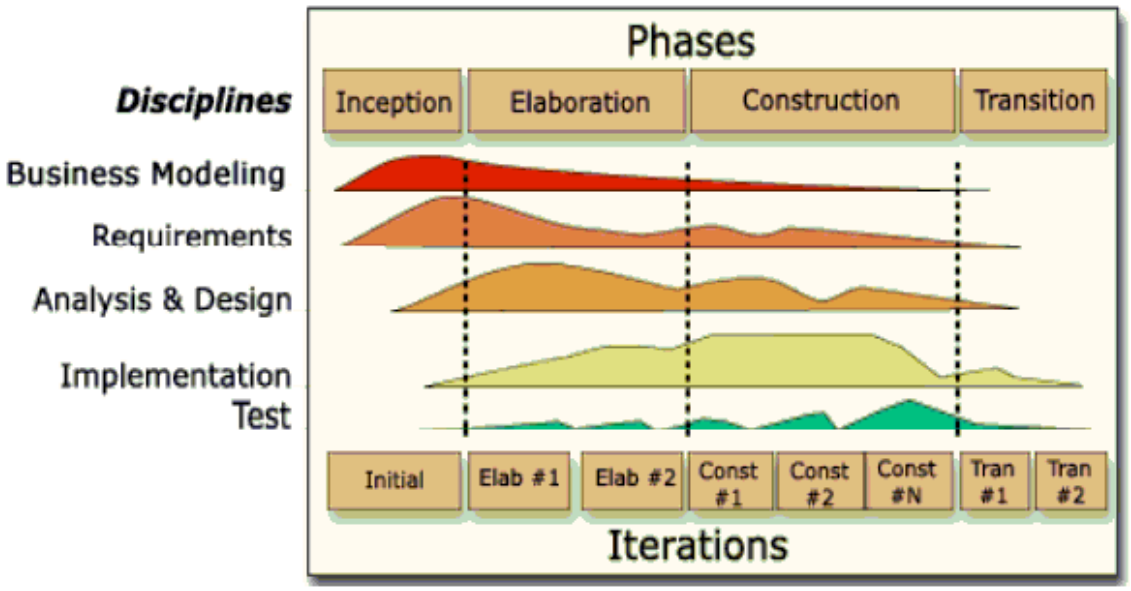


Figura 13: Flujos de Trabajo y Fases en el RUP

En otras palabras, el Proceso Unificado se repite a lo largo de una serie de ciclos que constituyen la vida de un sistema. Cada ciclo se desarrolla a lo largo del tiempo y concluye con una versión del producto que consta de las cuatro fases enunciadas anteriormente. Cada fase se subdivide a su vez en iteraciones. La figura 13 (extraída de [RUP]) muestra la representación de las fases y los flujos de trabajo o componentes del proceso.

3.2.1. Las cuatro fases

En la fase de *inicio*, se estudia la dinámica y estructura del negocio e identifican la mayoría de los casos de uso que permiten delimitar el sistema y el alcance del proyecto. Se identifican y priorizan los riesgos más importantes, se planifica en detalle la fase de elaboración, y se estima el proyecto de manera aproximada.

En la fase de *elaboración*, se especifican en detalle la mayoría de los casos de uso y se diseña la arquitectura del sistema. La arquitectura se expresa en forma de vistas de todos los modelos del sistema, que juntos representan al sistema entero. Esto implica que hay vistas arquitectónicas del modelo de casos de uso, del modelo de análisis, y del modelo de diseño.

Durante la fase de *construcción*, se crea el producto. La descripción evoluciona hasta convertirse en un producto preparado para ser entregado al usuario. Al final de

esta fase, el producto contiene todos los casos de uso que los desarrolladores y el cliente han acordado para el desarrollo de esta versión. Sin embargo, muchos defectos se descubren y solucionan en la fase siguiente.

La fase de *transición* cubre el período durante el cual el producto se convierte en versión beta. En la versión beta un número reducido de usuarios con experiencia prueba el producto e informa de defectos y deficiencias. Los desarrolladores corrigen los problemas e incorporan algunas de las mejoras sugeridas en una versión general dirigida a la totalidad de la comunidad de usuarios. El equipo de mantenimiento suele dividir los defectos en dos categorías: los que tienen suficiente impacto en la operación para justificar una versión incrementada y los que pueden corregirse en la siguiente versión normal.

3.2.2. Flujos de Trabajo

El *Modelado de Negocio* es una técnica que permite comprender los procesos de negocio. Su principal objetivo es estudiar y comprender la estructura y la dinámica de la organización sobre la cual se desarrollará el sistema.

La *Captura de Requerimientos* tiene como objetivo principal describir lo que el sistema hará y permitir a los desarrolladores y al cliente estar de acuerdo con dicha descripción, es decir, representar convenientemente los requerimientos para que desarrolladores, usuarios y clientes adquieran una comprensión común.

En el *Modelo de Análisis*, el objetivo es analizar los requisitos que se describieron en la captura de requerimientos, refinándolos y estructurándolos para conseguir una mayor comprensión de los mismos.

En el *Modelo de Diseño* se logra una arquitectura estable como base fundamental del modelo de implementación y se adquiere mayor compresión respecto a los requerimientos no funcionales, como el lenguaje de programación, sistema operativo, componentes de reuso, distribución y concurrencia y tecnologías de interfaz-usuario.

En el *Modelo de Implementación* se implementan los casos de uso en términos de componentes de código fuente, binarios y ejecutables.

El *Modelo de Prueba* se centra en las fases de elaboración, cuando se prueba la línea base ejecutable de la arquitectura, y de construcción, cuando la mayor parte del sistema esta implementada. Durante la fase de transición, la tarea de prueba está centrada en corregir los defectos y las pruebas de regresión.

A continuación y en el resto de este capítulo se exponen los artefactos y sus características del Modelo de Negocio. Una descripción más completa del RUP se encuentra en el anexo I de este trabajo.

3.3. Modelo de Negocio

3.3.1. La importancia del Modelado del Negocio

La figura 13 muestra que según la metodología del Proceso Unificado de Rational el primer flujo de trabajo a tener en cuenta es el modelado del negocio.

El modelado del negocio es una parte muy importante del ciclo de vida de desarrollo de software, que claramente ayuda a definir los requerimientos del sistema. El equipo de proyecto define los requerimientos del sistema a partir del análisis del problema a resolver en el contexto del negocio, de manera que esto ayude a asegurar que el sistema que se construirá se adecua a los objetivos del negocio. En efecto, el modelo de negocio puede ser una entrada directa al modelo de requerimientos del sistema y es el punto de partida fundamental para los siguientes modelos en el ciclo de vida del desarrollo del software.

El modelo de negocio no es necesario cuando las situaciones a resolver están claramente comprendidas, o cuando la organización no presenta ningún tipo de complejidad que justifique este modelo. En situaciones donde el desarrollo del sistema coincide con el comienzo de un nuevo negocio, la definición clara de los procesos a automatizar puede resultar un elemento de vital importancia que incide directamente en el éxito o fracaso de todo el negocio y del sistema mismo.

La finalidad del modelado del negocio es describir cada proceso del negocio, especificando sus datos, actividades (o tareas), roles (o agentes) y reglas de negocio. Y su propósito puede resumirse en las siguientes consideraciones:

- Establecer una abstracción de la organización.
- Entender la estructura y dinámica de la organización donde el sistema será desarrollado.
- Detectar y entender los reales problemas de la organización e identificar potenciales o posibles mejoras.
- Asegurar a clientes, usuarios, y desarrolladores que poseen una comprensión común de la organización.
- Derivar los requerimientos del sistema que son necesarios para representar satisfactoriamente la estructura y dinámica de la organización.

Básicamente, la propuesta de [RUP] se basa en lograr un buen entendimiento del negocio para la construcción de un sistema correcto. Esta construcción se realiza a partir de analizar los roles y responsabilidades de las personas y los elementos que participan del negocio. La figura 14 (extraída de [RUP]), muestra que los modelos que constituyen el Modelo del Negocio son: el modelo de casos de uso del negocio y el

modelo de objetos del negocio, llamado actualmente modelo de análisis del negocio, y a partir de ellos, se evoluciona a los restantes modelos que componen el ciclo de vida de desarrollo de un sistema.

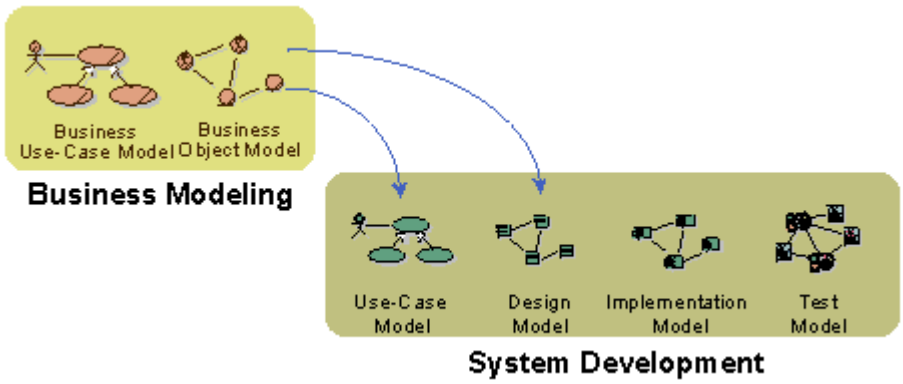


Figura 14: Del Modelo de Negocio a los Modelos del Sistema

Además, RUP propone el flujo de trabajo representado con un diagrama de actividades de UML en la figura 15 (extraída de [RUP]), el cual define el conjunto de actividades necesarias que permiten ejecutar una instanciación completa del modelo de negocio. El diagrama muestra que el propósito inicial es categorizar la organización en el cual el sistema se desarrollará (Assess Business Status), definir los escenarios más apropiados, tomar decisiones para continuar y definir metas y objetivos. Sobre dicha base se podrá tomar la decisión de continuar con las siguientes iteraciones y la forma de trabajar. En algunos casos, basta con definir el modelo de dominio, considerado como un subconjunto del modelo de objetos de negocio abarcando las entidades del negocio. Y cuando es necesario incluir cambios importantes en el negocio actual, se deben modelar ambos: el negocio actual y el nuevo negocio.

3.3.2. Artefactos del Modelo de Negocio

Artefacto es un término general para cualquier tipo de información creada, producida, cambiada o utilizada por los trabajadores en el desarrollo del sistema. El tipo de artefacto más importante en el Proceso Unificado es el *modelo*. Un modelo representa una perspectiva del sistema utilizada por algún trabajador. El Proceso Unificado sugiere comenzar el desarrollo de cualquier sistema a partir de la construcción del *modelo de negocio*, y la descripción más detallada de este modelo recaerá en la definición y uso de otros artefactos.

El *modelo de negocio* del RUP define un conjunto de artefactos agrupados por el responsable de realizarlos, como se muestra en la figura 16.

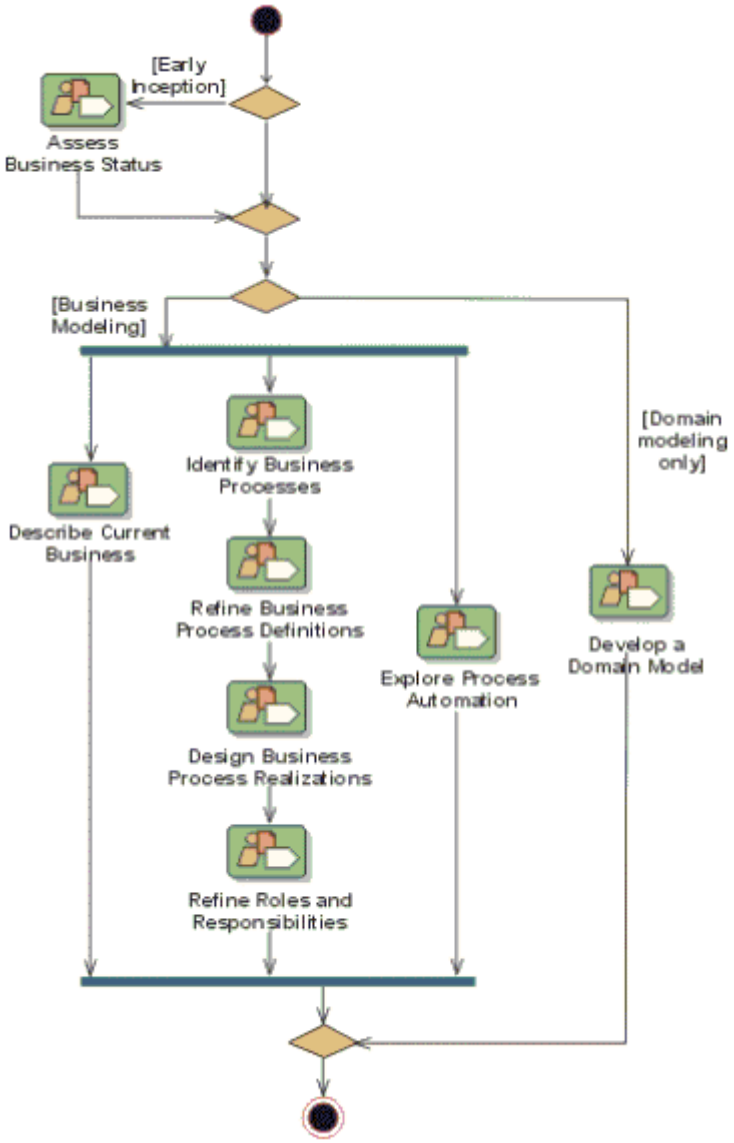


Figura 15: Workflow del Modelo de Negocio

El conjunto completo de artefactos del modelo de negocio capturan y presentan el contexto del sistema, sirven como entrada y referencia para la definición de los requerimientos del sistema.

Los dos artefactos fundamentales en el Modelo de Negocio son el *Modelo de Casos de Uso de Negocio* que describe un negocio en términos de Casos de Uso de Negocio y Actores de Negocio, y el *Modelo de Análisis de Negocio* que define la Realización de los Casos de Uso de Negocio. Los artefactos del modelo de negocio pueden agruparse en *modelos*, *elementos de modelos* o *documentos*. En el capítulo 4 se presenta esta categorización en detalle junto al modelo genérico propuesto.

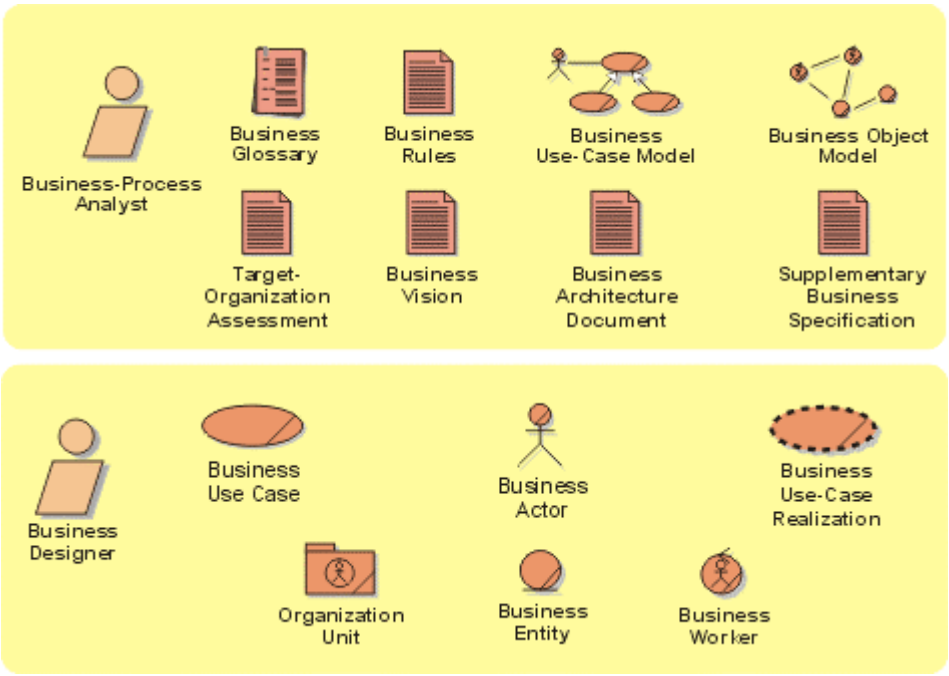


Figura 16: Artefactos del modelo de negocio definido por el RUP

A continuación se describen brevemente cada uno de los artefactos que componen el modelo de negocio según la propuesta realizada por Rational Unified Process [RUP].

3.3.2.1. Artefacto: Glosario de Negocio (Business Glossary)

El propósito del glosario del negocio es definir los términos específicos del proyecto que usuarios, analistas y desarrolladores deben entender y usar. En otras palabras, el glosario de negocio define el vocabulario común usando los términos y expresiones más comunes del dominio del problema.

Es responsabilidad del analista del proceso de negocio preservar la integridad del glosario, asegurando que éste es producido temprana y oportunamente y que mantiene consistencia con los resultados del desarrollo.

El Glosario se produce primariamente, durante la fase de inicio. Se debe usar consistentemente este vocabulario común en todas las descripciones textuales del negocio. Mantener las descripciones textuales consistentes evita malos entendidos entre los miembros del proyecto acerca del uso y significado de los términos.

Para focalizar sobre los términos que definen el glosario del negocio se deben describir los siguientes conceptos:

- Objetos del Negocio: representando conceptos usados en el trabajo diario de la organización.

- Objetos de Mundo real: aquellos que el negocio necesita conocer, que ocurren naturalmente e incluyen cosas como: casa, perro, factura, etc.

Generalmente, cada término se describe con un sustantivo y una descripción textual que lo define.

3.3.2.2. Artefacto: Reglas de Negocio (Business Rules)

Las reglas de negocio definen una restricción o invariante que el negocio debe satisfacer. Dichas restricciones pueden ser de comportamiento o estructurales.

Estas reglas deberían ser rigurosamente formalizadas ya que pueden representar una base para la automatización. Pero por razones de costo y recursos, normalmente se describen en lenguaje natural o en el lenguaje de restricción de objetos OCL (Object Constraint Language) [OCL01].

Se detectan primariamente durante la fase de inicio, y son detalladas durante las fases de elaboración y construcción

Las reglas de negocio se dividen en dos categorías: reglas de Restricción y regla de Derivación.

Reglas de Restricción

Especifican políticas o condiciones que restringen el comportamiento y la estructura del objeto, y son:

- Reglas de Estímulo y Respuesta: restringen el comportamiento especificando que cuando una condición es verdadera el comportamiento será ejecutado. Pueden afectar el flujo de eventos que describe un caso de uso de negocio, y aparecer como actividades específicas o como caminos condicionales o alternativos que restringen el flujo. Además, pueden afectar la descripción de una entidad de negocio o ser parte de la descripción de una operación llevada a cabo por un trabajador (worker) del negocio. Por ejemplo:

  "Una Factura puede ser cancelada solo si no fue enviada al Cliente"

- Reglas de Restricción de Operación: se traducen generalmente como pre y poscondiciones del flujo de eventos de un caso de uso de negocio que aseguran el mismo se ejecuta correctamente. Un ejemplo puede ser:

  "Enviar la Factura al Cliente solo si este posee una dirección de envío"

- Reglas de Restricción de Estructura: especifican políticas o condiciones respecto de clases, objetos y sus relaciones que no deben ser violadas en ningún momento. Por ejemplo, dadas dos entidades FacturaVenta y Producto, una regla de restricción de estructura puede ser:

  " Una FacturaVenta contiene al menos un Producto"

Reglas de Derivación

Especifican políticas o condiciones para deducir o computar hechos a partir de otros hechos, y son:

- Reglas de Inferencia: especifican que si ciertos hechos se cumplen una conclusión puede ser inferida. Estas reglas implican un método normalmente reflejado en algún estado de actividad del flujo de eventos (workflow) de un caso de uso de negocio y eventualmente sobre alguna operación de una entidad de negocio o de un trabajador (worker) de negocio. Un ejemplo para este tipo de reglas puede ser la siguiente afirmación:

  "Un cliente es buen cliente sii pagó siempre a término"

- Reglas de Computación: representan un método a menudo expresado en un algoritmo. Dicho método puede ser reflejado como una actividad en el flujo de eventos (workflow) de un caso de uso de negocio o como una operación sobre un trabajador (worker) de negocio o una entidad de negocio. Un ejemplo de una regla de computación puede ser el siguiente:
  *"El precio del producto es computado como: precioProducto * (1+ iva/100)"*

3.3.2.3. Artefacto: Modelo de Casos de Uso de Negocio (Business Use Case Model)

El modelo de casos de uso del negocio representa la funcionalidad del negocio. Es usado como entrada fundamental para identificar los roles en la organización y describe los procesos de negocio de una empresa en términos de casos de uso del negocio y actores del negocio.

A partir de la construcción de este modelo, se logra un mejor entendimiento del contexto del negocio por parte de todos los involucrados: cliente, arquitecto de software, analista de sistemas, encargado de la planificación, etc.

Es responsabilidad del analista del proceso de negocio asegurar que el modelo de casos de uso del negocio sea consistente, correcto y legible, y que cubre suficientemente el negocio para proveer una base sólida para la construcción del sistema.

Se produce parcialmente durante la fase de inicio y totalmente en la fase de elaboración. Si el negocio se reestructura es necesario confeccionar dos modelos de negocio, el actual y el nuevo negocio. El costo de estos dos modelos es importante, por lo tanto se debe ser muy cuidadoso al considerar hacerlos.

3.3.2.4. Artefacto: Caso de Uso de Negocio (Business Use Case)

Un caso de uso de negocio es una secuencia de acciones que el negocio desarrolla para llegar a un resultado observable por un actor del negocio particular.

El propósito de un Caso de Uso de Negocio es describir un proceso de negocio desde un punto de vista del valor agregado externo. Los casos de uso de negocio son útiles para conocer el valor que provee el negocio y la manera en que interactúa con su ambiente.

Los stakeholders, analistas del proceso y diseñadores del negocio usan los casos de uso de negocio para describir los procesos del negocio y para entender el efecto de cualquier cambio propuesto a la forma en que el negocio funciona en la actualidad. Los casos de uso de negocio también son usados por los analistas del sistema y los arquitectos del software para entender la forma en que el sistema se ubicará en la organización. Los gerentes de prueba los usan para proveer un contexto y desarrollar escenarios de prueba. Los gerentes del proyecto los usan para planificar el contexto de las iteraciones del modelado del negocio.

Es responsabilidad del analista del proceso del negocio garantizar la integridad de los casos de uso de negocio, asegurando que:

- Describen correctamente la forma en que la organización realiza su negocio.
- La secuencia de acciones del caso de uso de negocio es legible y cumple su propósito.
- Las relaciones de include y extend[4] originadas entre los casos de uso de negocio son justificadas y se mantienen consistentes.
- El rol del caso de uso de negocio donde está involucrado (en relaciones de asociación-comunicación) es claro e intuitivo.
- Los diagramas que describen los casos de uso de negocio y sus relaciones son legibles y cumplen su propósito.
- Los requerimientos especiales son legibles y cumplen su propósito.
- Las pre-condiciones y las post-condiciones son legibles y cumplen su propósito.

Los casos de uso de negocio se identifican y describen brevemente en la fase de inicio para ayudar a definir el alcance del proyecto. Si el modelado del negocio se hace como parte de una re-ingeniería del negocio, entonces los casos de uso de negocio arquitectónicamente más significativos serán detallados durante la fase de elaboración y los restantes en la fase de construcción. En cambio, si el modelado del negocio se

hace como parte del desarrollo de un sistema de software, los casos de uso de negocio son descriptos en detalle en la fase de elaboración.

Cada caso de uso de negocio debe soportar al menos un objetivo de negocio. Las estrategias del negocio se trasladan a objetivos del negocio concretos y medibles, y pueden ser soportados por los procesos o casos de uso de negocio. Definir claramente la relación entre objetivos del negocio y los casos de uso de negocio, asegura que los procesos de negocio se alinean con las estrategias del negocio. La existencia de esta relación ayuda a priorizar y factorizar los casos de uso de negocio.

3.3.2.5. Artefacto: Actor del Negocio (Business Actor).

Para entender el propósito del negocio se deben conocer quiénes interactúan con él, es decir, quiénes hacen demandas o están interesados en sus salidas.

Un rol que alguien o algo juega cuando interactúa con el negocio se llama actor de negocio. Un actor de negocio representa un tipo particular de usuario del negocio y no un usuario físico real. El nombre unívoco que identifica al actor de negocio debería reflejar su rol en el negocio.

Los analistas del sistema del negocio utilizan este artefacto cuando definen los límites de la organización. Los diseñadores del negocio lo usan cuando describen los casos de uso de negocio y su interacción con los actores. Los diseñadores de la interfaz de usuario lo usan como entrada para capturar las características de los actores humanos. Los analistas del sistema lo usan como entrada para encontrar los actores del sistema.

Los analistas de los procesos del negocio son responsables de la integridad de los actores de negocio asegurando que:

- Cada actor del negocio humano captura las características necesarias.
- Cada actor del negocio tiene la asociación correcta con el caso de uso de negocio en el que participa.
- Cada actor del negocio es parte de una relación de generalización correcta.
- Cada actor del negocio define un rol cohesivo, y es independiente de los otros actores del negocio.

Los actores del negocio son encontrados y relacionados a los casos de uso de negocio en la fase de inicio. También es posible relación de generalización para representar una relación entre actores del negocio.

[4] Include: un caso de uso incluye explícitamente el comportamiento de otro. Extend: un caso de uso extiende su comportamiento a través de otro cuando se cumplen ciertas condiciones.

3.3.2.6. Artefacto: Objetivo de Negocio (Business Goal).

El propósito de los objetivos de negocio es trasladar las estrategias de negocio a una forma concreta y mensurable, de manera que las operaciones del negocio se puedan llevar por la dirección correcta y mejorar si es necesario. Estas medidas son cuantitativas.

Los gerentes del negocio y los stakeholders usan los objetivos de negocio para asociar las estrategias del negocio a medidas concretas. Los analistas y diseñadores los usan para verificar que los procesos del negocio están asociados a las estrategias del negocio.

Se construyen en la fase de inicio y se usan como entrada al modelo de casos de uso de negocio.

Una organización tiene una visión del negocio, la cual se traduce a estrategias del negocio. Estas estrategias deberían ser resumidas por los objetivos de negocio que finalmente son medidos en las operaciones de la organización. La visión del negocio es implementada por los actores de negocio y workers de negocio, interactuando para la realización de casos de uso de negocio.

Estrategias de Negocio y Objetivos de Negocio

Las estrategias de negocio definen la manera en la cual la organización debería interactuar con su ambiente. Se focalizan en una perspectiva externa a la organización.

Los objetivos de negocio definen lo que se debería lograr para conseguir una posición de la organización competitiva y sostenible. Definen lo que se necesita conseguir para realizar un objetivo de alto nivel, mientras que las estrategias proveen los límites dentro de los cuales estos objetivos estarán definidos.

3.3.2.7. Artefacto: Modelo de Análisis de Negocio (Business Analisys Model)

El modelo de análisis de negocio es una abstracción que describe la forma en que cada caso de uso de negocio es llevado a cabo por parte de un conjunto de trabajadores de negocio que utilizan las entidades del negocio y las unidades de trabajo, normalmente llamado realización de los casos de uso de negocio. Cada realización puede representarse gráficamente con diagramas de interacción y diagramas de actividades de UML. En los diagramas de actividades, las calles corresponden a elementos de objetos de negocio.

El modelo de análisis de negocio, define los casos de uso de negocio desde el punto de vista interno de los trabajadores de negocio. El modelo define la forma de

trabajo de las personas, qué manejan y usan, y cómo están relacionados estática y dinámicamente para producir los resultados esperados.

Una entidad de negocio representa algo que los trabajadores toman, inspeccionan, manipulan, producen o utilizan en un caso de uso de negocio. Son equivalentes a las clases del dominio en un modelo de dominio. Una unidad de trabajo representa un conjunto de entidades. Cada trabajador, entidad y unidad de trabajo pueden participar en más de una realización de caso de uso del negocio.

El modelo de análisis de negocio es muy usado como entrada para identificar los actores y los casos de uso del sistema.

Modelo de Dominio

El Modelo de Dominio es un subconjunto del Modelo de Análisis de Negocio que no incluye las responsabilidades. El objetivo del modelado del dominio es comprender y describir las clases más importantes dentro del contexto del sistema.

Hay clases, que si bien son candidatas, no son incluidas en el modelo de dominio y forman el glosario de términos. Cuando el dominio de negocio es muy pequeño, no es necesario definir el modelo de dominio y solo se define el glosario.

Ambos, el glosario y el modelo de dominio, ayudan a todos los participantes del proyecto a lograr un vocabulario común y un entendimiento coordinado y profundo del problema a resolver.

Es fundamental comprender que el objeto del modelo de dominio es ayudar a definir lo que hará el sistema para resolver el problema, y no la manera en que lo hará.

Se utiliza en el modelo de casos de uso para identificar los casos de uso del sistema y en el modelo de análisis para identificar clases entidad.

El modelo de dominio es representado gráficamente con un diagrama de clases de UML.

3.3.2.8. Artefacto: Realización de Caso de Uso de Negocio (Business Use Case Realization).

Una realización de caso de uso de negocio describe como los trabajadores, entidades, y eventos de negocio colaboran para ejecutar un caso de uso de negocio particular. En otras palabras, la realización de casos de uso de negocio muestra los detalles de cómo los procesos de negocio son ejecutados. La estructura de dichas realizaciones sigue el flujo en los casos de uso de negocio y definen quién está haciendo qué dentro de la organización.

Este artefacto es parte del modelo de análisis de negocio y debe ser modelado si el flujo de eventos organizacional es muy importante o si los potenciales cambios pueden afectar la forma en que opera el negocio actual.

Mientras que un caso de uso de negocio es descrito desde una perspectiva externa, con la intención de describir los pasos que serán ejecutados para dar valor a algún integrante del proyecto, la realización de caso de uso de negocio describe la manera en que esos pasos se ejecutarán dentro de la organización, desde una perspectiva interna.

Este artefacto es usado para asegurar que todos los miembros del equipo de proyecto comprenden la manera de operar del negocio.

Las realizaciones de caso de uso de negocio son definidas y priorizadas principalmente durante la fase de inicio. Los detalles de alta prioridad serán realizados en la fase de inicio y elaboración, y el resto se realizará en la fase de construcción.

Un diagrama de actividades de UML es apropiado para modelar el flujo de control que una realización de caso de uso de negocio representa, facilitando el descubrimiento de las responsabilidades que corresponden a cada trabajador del negocio. Además, los diagramas de actividades permiten detectar con claridad el envío y recepción de eventos entre trabajadores del negocio. Y a partir de un minucioso análisis de las actividades representadas en este tipo de diagramas se extrae una importante cantidad de casos de uso del sistema.

Cuando este artefacto es excluido, cualquier requerimiento suplementario se incluye en el artefacto especificaciones suplementarias del negocio.

3.3.2.9. Artefacto: Sistema de Negocio (Business System)

Un sistema de negocio encapsula un conjunto de roles y recursos que juntos completan un propósito específico, y define las responsabilidades a través de las cuales el propósito puede conseguirse.

Los sistemas de negocio tienen el propósito fundamental de reducir la complejidad de interdependencias e interacciones dentro del negocio. Esto se hace definiendo un conjunto de capacidades de manera que las dependencias sobre estas capacidades no necesiten tener conocimiento de la forma en que son desarrolladas. De esta forma los sistemas de negocio son usados de la misma manera que los componentes de software y hardware, porque definen una unidad de estructura que encapsula los elementos estructurales que contiene y son caracterizados por propiedades visibles externamente.

Este artefacto forma parte del modelo de análisis de negocio.

Los sistemas de negocio son usados por los analistas del proceso del negocio para determinar si las capacidades requeridas dentro de la organización están presentes y para asegurar que el modelo del negocio está anticipando el cambio o al menos es flexible al cambio. Los diseñadores del negocio usan los sistemas de negocio para formar colecciones de trabajadores y entidades relacionados y definir explícitamente dependencias dentro de la organización. Los gerentes del proyecto también los usan para planificar trabajos en paralelo.

Los sistemas de negocio y sus capacidades se identifican durante la fase de inicio, sus elementos contenidos y responsabilidades se detallan durante la fase de elaboración.

3.3.2.10. Artefacto: Entidad de Negocio (Business Entity)

Una entidad de negocio representa una pieza de información persistente y significante que es manipulada por actores y trabajadores de negocio.

Las entidades son pasivas, es decir, ellas no inician interacciones. Una entidad puede usarse en diferentes realizaciones de caso de uso de negocio. Las entidades proveen la base para compartir información entre los trabajadores que participan en las diferentes realizaciones.

Este artefacto forma parte del modelo de análisis de negocio.

Los miembros del proyecto usan las entidades de negocio para asegurar que toda la información creada y requerida por la organización esta presente en el modelo de análisis de negocio. Las entidades son usadas por los analistas y diseñadores del sistema para describir los casos de uso del sistema e identificar las entidades de software.

Las entidades más significativas se identifican durante la fase de inicio. Las restantes durante la fase de elaboración. El conjunto de entidades del negocio forman el modelo de dominio mencionado anteriormente.

3.3.2.11. Artefacto: Trabajador de Negocio (Business Worker)

Un trabajador de negocio es una abstracción de un humano o sistema de software, que representa un rol desarrollado dentro de las realizaciones de caso de uso de negocio. Esta abstracción permite identificar mejoras potenciales en los procesos del negocio.

Un trabajador de negocio colabora con otros, es notificado de eventos de negocio y manipula entidades para desarrollar sus responsabilidades.

Este artefacto forma parte del modelo de análisis de negocio.

Los miembros del proyecto usan este artefacto para confirmar que la asignación de responsabilidades e interacciones reflejan correctamente la forma de trabajo actual del negocio, y les permite detectar problemas o fallas que pueden ser mejoradas. Además, los trabajadores se usan para considerar el impacto de cualquier cambio en la organización, como los es la automatización de los procesos del negocio. El diseñador del negocio describe las realizaciones de casos de uso de negocio usando las responsabilidades asignadas a los trabajadores. Además, este artefacto le sirve a los analistas del sistema para identificar actores y casos de uso del sistema.

Los trabajadores de negocio son definidos en la fase de inicio y detallados y refinados en la fase de elaboración.

3.3.2.12. Artefacto: Evento de Negocio (Business Event)

Los eventos de negocio representan hechos importantes que suceden en el negocio y ayudan a manejar la complejidad. Son disparados y recibidos por los actores, trabajadores y entidades de negocio mientras interactúan para la realización de casos de uso. Los eventos se usan para disparar casos de uso del negocio, para señalar cambios de estado del negocio, y para información entre casos de uso del negocio.

Este artefacto forma parte del modelo de análisis de negocio.

Los miembros del proyecto del negocio los usan para una mejor comprensión y descripción de las actividades del negocio. Los diseñadores son responsables de identificar y detallar los eventos. Y son usados por los analistas del sistema para identificar o refinar actores y casos de uso del sistema.

3.3.2.13. Artefacto: Evaluar el estado corriente de la Organización (Target Organization Assessment)

Describe el estado corriente de la organización en el cual se desarrolla el sistema. Dicha descripción se realiza en términos de procesos corrientes, herramientas, competencia, actividades, clientes, competidores, tendencias técnicas, problemas y áreas de mejoramiento.

Este artefacto es usado por el analista de procesos de negocio como una base para la configuración de la disciplina del modelo de casos de uso de negocio para un proyecto particular. También es usado para explicar a los miembros del proyecto la necesidad de cambiar ciertos procesos de negocio. Permite crear motivación y un entendimiento común entre las personas de la organización. Además, ayuda a comprender mejor las necesidades del cliente, ver el funcionamiento de otros

negocios, la intención de los dueños, las necesidades de los empleados y las nuevas tecnologías.

Este artefacto es creado al inicio del proyecto.

Ideas y Estrategias del negocio

La idea del negocio es identificar los productos y servicios que la compañía quiere ofrecer y los mercados donde toman lugar.

La estrategia del negocio define los principios de la forma en que la idea debe ser lograda y define los objetivos a largo plazo.

La estrategia de negocio debe estar de acuerdo con la forma que trabaja el negocio. Debe incluir los objetivos a largo plazo y alinear los casos de uso del negocio con los objetivos del negocio. Un caso de uso del negocio correcto sigue la dirección de la estrategia.

Los criterios a tener en cuenta para definir una buena estrategia pueden ser:

- No debe estar basada solo en objetivos financieros.
- Debe ser formulada de manera que sus efectos puedan ser medidos.
- Debe enfocar sobre una idea de negocio delimitada y real.

Generalmente, se utiliza el Benchmarking como técnica para el análisis de información e intercambio de conocimientos con otros negocios.

3.3.2.14. Artefacto: Visión del Negocio (Business Vision)

La visión del negocio define el conjunto de objetivos a los cuales el modelo de negocio debe enfocar. Captura los objetivos de alto nivel de un modelo de negocio. Provee una entrada al proceso de aprobación del proyecto. Comunica los *Qué´s* y los *Cómo´s* fundamentales del proyecto y es una medida por el cual las futuras decisiones serán validadas.

3.3.2.15. Artefacto: Especificaciones Suplementarias del Negocio (Suplementary Business Specification)

Este artefacto presenta cuantificadores del negocio o restricciones que el negocio debe cumplir, que no están en el modelo de casos de uso del negocio ni en el modelo de análisis del negocio.

Este documento captura descripciones de procesos, cuantificadores o restricciones que pueden ser aplicables a todos los casos de uso del negocio.

Es utilizado por los miembros del proyecto para entender los efectos de los cuantificadores y restricciones, y capturar los requerimientos de software.

Este documento no debe contener objetivos del negocio, dado que éstos son identificados, analizados y modelados por separado. Los objetivos del negocio son usados para planear y dirigir las actividades del negocio en la dirección de la estrategia de negocio, mientras que las especificaciones suplementarias del negocio son usadas para definir los límites entre los cuales los negocios operan.

Este artefacto es desarrolla durante las fases de inicio y elaboración.

En el próximo capítulo, se presenta el modelo genérico obtenido a partir del análisis de los artefactos que componen el modelo de negocio, anteriormente descriptos.

CAPITULO 4: Modelo Genérico del Modelo de Negocio

4.1. Introducción

Para el abordaje al diseño y desarrollo de un sistema de software es necesario estudiar en detalle la estructura y dinámica de la organización donde el mismo funcionará, con el objeto de obtener los requerimientos del sistema más correctamente.

Entender la estructura y dinámica de la organización, detectar los problemas corrientes e identificar potenciales o posibles mejoras y asegurar un entendimiento común de la organización entre todos los participantes, son algunos de los focos principales a la hora de decidir el diseño y desarrollo de un sistema de software.

El propósito del desarrollo de un sistema de software es solucionar problemas a través de programas eficientes, robustos, seguros, dinámicos, interactivos, portables, entre otras características. Para lograr este propósito, es necesario y fundamental estudiar y analizar el problema en distintos niveles de detalle y proponer las posibles soluciones.

En el capítulo 3 se introdujeron las características principales del Proceso Unificado de desarrollo de software [RUP] [JBR99], y en particular se describieron en detalle los artefactos propuestos para el modelado del negocio. Este modelo es propuesto en la primera etapa del proceso con el objeto de establecer una clara abstracción de la organización, logrando un íntegro entendimiento del negocio para la construcción de un sistema correcto. El modelo de negocio del Proceso Unificado está compuesto por un conjunto de artefactos (Figura 16) que permiten concretar el objetivo propuesto con la construcción de dicho modelo.

Los trabajos existentes acerca del Modelado de Negocio y que hacen referencia específicamente a la metodología del Proceso Unificado, utilizan la definición de los artefactos, tal como la presentan sus autores, sin mayor análisis. En este plano cabe mencionar trabajos como el libro escrito por Eriksson y Penker [EP00] donde definen una práctica extensión a UML para describir procesos de negocio, y ponen a disposición una galería de patrones de negocio para su aplicación en casos reales. Presentan conceptos claves del modelado de negocio, e incluyen una descripción de cómo definir reglas de negocio en Object Constraint Language (OCL) y como usar procesos de negocio como casos de uso del sistema. En [OMMN01] se propone un método sistemático para obtener el modelo de requisitos a partir del modelo de

negocio, donde una vez identificados los procesos de negocio de la organización, y descritos sus flujos de trabajo mediante diagramas de actividades de UML, los casos de uso se detectan y estructuran a partir de las actividades de cada proceso, mientras que las entidades del modelo conceptual se obtienen de los datos que fluyen entre tales actividades. Además, se identifican las reglas del negocio y se incluyen en un glosario como parte de la especificación de los datos y las actividades. Nada se indica del resto de los artefactos. En [Dapena02] se propone una técnica para desarrollar sistemas utilizando el Razonamiento Basado en Casos (RBC), y obtener el Modelo del Negocio a partir de un conjunto de especificaciones iniciales brindadas por los analistas, permitiendo aprovechar la experiencia de otros desarrolladores en el diseño de sistemas con características similares. [Salm03] utiliza las extensiones a UML para el Modelado de Negocio propuestas por la OMG. En particular se utilizan los estereotipos, que son capaces de contemplar una visión inicial de los procesos de negocio, siendo posible capturar de forma significativa eventos, entradas, recursos y salidas asociadas a un proceso de negocio. [SVC01] proponen una extensión a UML con la definición de un Profile para el modelo de negocio donde específicamente definen los procesos de negocio, su relación con los objetivos y los recursos. [Heumann03] destaca la importancia del modelado del negocio en el ciclo de vida de desarrollo de un sistema, usando un lenguaje de modelado visual como UML. Introduce con claridad los dos modelos a construir en el modelado de negocio, el modelo de casos de uso de negocio y el modelo de objetos de negocio o modelo de análisis de negocio, y sugiere los diagramas de UML a usar de acuerdo a cada representación.

Ninguno de los trabajos existentes facilita la tarea del desarrollador, reduciendo tiempos en la etapa de comprensión del contexto, y por consiguiente reduciendo considerablemente los costos de esta etapa, y de todo el proyecto, porque como ya se mencionó, obtener un estudio claro y seguro del contexto resulta en la obtención de una lista de requerimientos más correcta, y por lo tanto minimiza los errores en las siguientes etapas, siempre apuntando a la concreción de un software de calidad.

Partiendo del estudio y el análisis de cada uno de los artefactos que componen el modelo de negocio, se construye el modelo genérico del modelo de negocio que organiza y muestra las relaciones detectadas entre artefactos y define un conjunto de reglas que el modelo debe cumplir. El modelo genérico se presenta gráficamente en términos de UML a través de un diagrama de clases, y podrá ser instanciado con diferentes modelos reales.

El modelo genérico propuesto, establece las bases para especificar un modelo concreto, y su propósito principal es facilitar al desarrollador la comprensión y

desarrollo del contexto del sistema. Con la aplicación de las reglas definidas para cada artefacto y la relación entre los mismos, el modelo genérico ayuda a construir un modelo suficiente y correcto para el conocimiento que se requiere del contexto. Recíprocamente, el modelo genérico puede ser también usado para asegurar que un modelo de negocio concreto previamente construido cumple con las especificaciones impuestas por la metodología.

El modelo genérico no modifica la estructura interna de cada artefacto propuesto por el RUP para el modelo de negocio, simplemente los organiza de acuerdo a las relaciones y reglas existentes entre los mismos, y los representa en el lenguaje gráfico UML que permite especificar el modelo en una forma que facilita la comunicación, visualización y entendimiento del desarrollador. El caso de estudio desarrollado en 4.4. sugiere un orden de construcción de los artefactos que sigue las recomendaciones originales del método, pero además considera las características impuestas por el modelo genérico presentado en este trabajo.

4.2. Modelo Genérico del Modelo de Negocio

Tal como lo postulan los creadores del proceso [JBR99], con las particularidades introducidas por [RUP], el modelo de negocio está soportado por dos artefactos principales, el modelo de casos de uso del negocio y el modelo de análisis del negocio.

El modelo de casos de uso de negocio describe los procesos de negocio de una empresa en términos de casos de uso del negocio y actores del negocio que se corresponden con los procesos del negocio y los clientes, respectivamente. Presenta un sistema desde la perspectiva de su uso, y esquematiza como proporciona valor a sus usuarios. Por otro lado, el modelo de análisis del negocio es un modelo interno a un negocio, que describe cómo cada caso de uso de negocio es llevado a cabo por un grupo de trabajadores que utilizan entidades del negocio y unidades de trabajo. El conjunto completo de artefactos del modelo de negocio captura y presenta el contexto del sistema, y sirven como entrada y referencia para los requerimientos del sistema.

El modelo genérico del modelo de negocio es una reconstrucción racional y una generalización (en rigor, un metamodelo) del modelo original presentado por el [RUP]. Permite visualizar y especificar de manera precisa, no ambigua y completa los artefactos que componen el modelo de negocio y sus relaciones y representar el modelo de negocio de organizaciones de cualquier ámbito.

La propuesta original utiliza el lenguaje natural para describir los artefactos y sus relaciones, y realiza una extensa descripción de cada uno. Menciona las

particularidades de cada artefacto, e impone relaciones entre los mismos. Es fundamental la necesidad de organizar esta información, con el objeto de facilitar la comprensión en la construcción de cada artefacto y reducir riesgos como imprecisión, ambigüedad, etc., muy peligrosos a la hora de traducirlos a una notación formal o semiformal (como por ejemplo los diagramas típicos de UML), donde necesariamente, el diseñador deberá interpretar, agregar y/o quitar información para poder llevar a cabo su tarea.

El modelo genérico propuesto simplifica la obtención de una solución al problema de modelado de negocio, ya que resume en un único diagrama de clases todos los artefactos que componen el modelo de negocio, y las relaciones entre ellos. Y por la propia semántica del diagrama de clases de UML (tipo de relaciones, multiplicidad, navegabilidad, etc.) el desarrollador puede visualizar con claridad los artefactos obligatorios de construir y los opcionales. Se denomina “artefactos obligatorios de construir” a aquellos que son necesarios en cualquier contexto y tamaño de organización, y que aseguren que la construcción del sistema se hará sobre una base mínima requerida de conocimiento del negocio.

Por otro lado, el RUP no especifica de forma explícita, algunas condiciones que se deben cumplir cuando se construye un modelo de negocio concreto. En este trabajo, se definen también un conjunto de reglas que acompañan al modelo genérico representado con el diagrama de clases UML, y tienen el objeto de ayudar a construir un modelo consistente, siempre que sean tenidas en cuenta durante el proceso de construcción del modelo de negocio específico. Además, estas reglas deben ser tenidas en cuenta cuando se trata de analizar la consistencia de un modelo de negocio previamente construido.

El modelo genérico y sus reglas adicionales, simplifican el esfuerzo del desarrollador para modelar un negocio correctamente, ya que le impone la obligación de realizar un análisis de las relaciones entre los artefactos del modelo de negocio sobre un lenguaje gráfico estándar y universal, fácil de aprender, interpretar y usar. Le ayuda a definir qué artefactos construir y en qué orden, y qué artefactos son convenientes descartar para cada caso particular. Además, el modelo genérico asegura que si la construcción de un modelo de negocio particular se basa en su definición y reglas, aumentan considerablemente las probabilidades de desarrollar un software de mayor calidad, que tiene un punto de partida que colabora en la obtención de resultados correctos y satisfactorios y favorecen el éxito de las siguientes etapas del ciclo de desarrollo, y por lo tanto el éxito del proyecto como un todo.

Para la construcción del modelo genérico, cada artefacto del modelo de negocio es definido como una clase, y por simplicidad es nombrada con una sigla que la identifica, como se indica en la tabla 4.1.

MODELOS	
MCUN	Modelo de Casos de Uso de Negocio
MAN	Modelo de Análisis de Negocio

DOCUMENTOS		ELEMENTOS DE MODELOS	
GN	Glosario de Negocio	**CUN**	Caso de Uso de Negocio
VFO	Valoración del Fin del Negocio	**AN**	Actor de Negocio
VN	Visión del Negocio	**RCUN**	Realización de CUN
DAN	Doc. Arquitectura del Negocio	**SN**	Sistema de Negocio
ESN	Esp. Suplementaria del Negocio	**EN**	Entidad de Negocio
RN	Reglas del Negocio	**WN**	Worker de Negocio
ON	Objetivo del Negocio	**EvN**	Evento de Negocio

Tabla 4.1. Siglas que identifican los Artefactos del Modelo de Negocio

En el capítulo 3 se describieron cada uno de estos artefactos en términos de la definición dada por sus autores, y en la figura 17 se muestra el Diagrama de Clases que representa el modelo genérico para el modelo de negocio propuesto en este trabajo.

Este diagrama de clases permite visualizar con claridad los artefactos que componen el modelo completo de negocio, destacando los *modelos* con el estereotipo <<model>>, *elementos de modelos* estereotipados como <<element>> y expresados con relaciones de composición, y *documentos* con el estereotipo <<document>>, que representan las definiciones suplementarias para los dos modelos fundamentales representados por MAN y MCUN.

4.2.1. Análisis del Modelo Génerico

A continuación se expone el análisis del modelo genérico del modelo de negocio que surge de la lectura del diagrama de clases de la figura 17 y de condiciones impuestas originalmente por el método.

- El MCUN es el modelo más importante a construir en el modelado de negocio. Si se analiza en detalle el MCUN y su relación con los demás artefactos, el diagrama de clases de la figura 17 especifica de manera precisa que para cada MCUN debe existir una instancia de los artefactos GN, RN y ON.

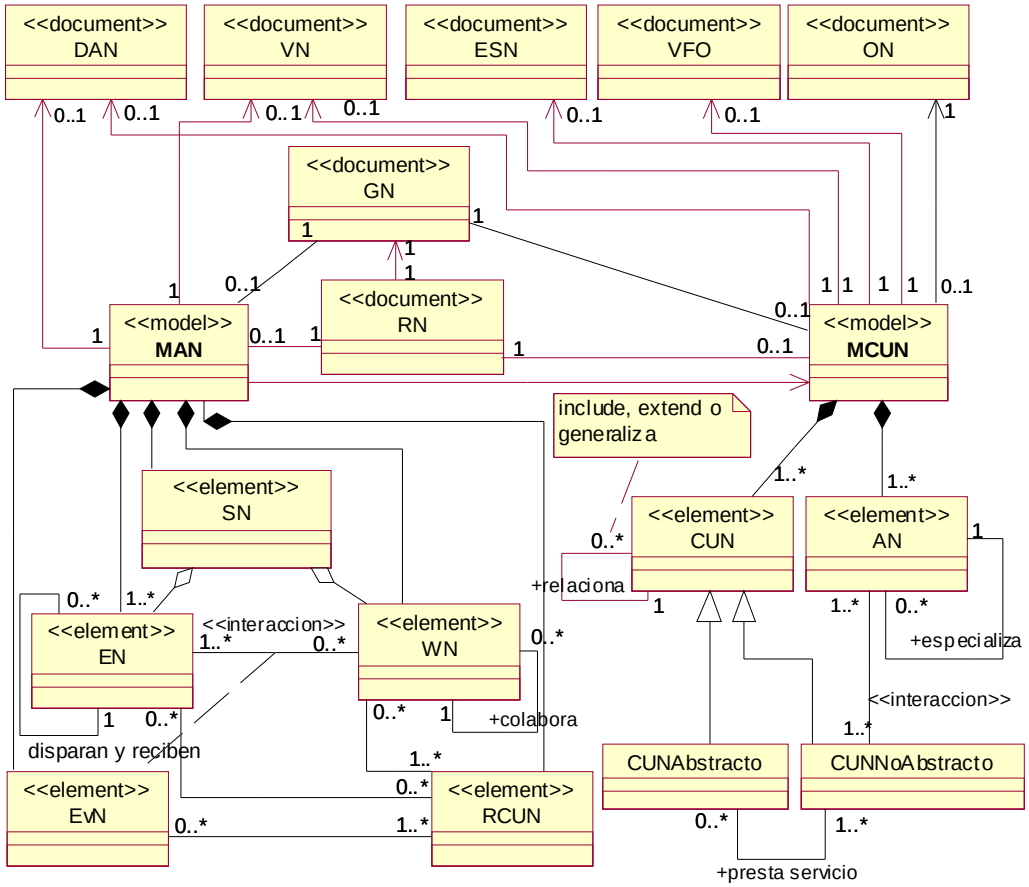


Figura 17: Modelo Genérico del Modelo de Negocio

- Es simple deducir que para crear el MCUN será necesario haber introducido las estrategias de negocio en ON, y una vez definidos, el MCUN dirige su propósito hacia ellos. Observando el diagrama, la relación entre MCUN y ON es una asociación simple con dirección hacia el ON con multiplicidad igual a 1 y con 0..1 en MCUN, que además del significado que representa la relación propiamente dicha, ésta pretende expresar que se requiere la definición de los objetivos del negocio (ON), al menos de manera parcial, antes de comenzar la descripción del modelo de casos de uso del negocio (MCUN) dado que su propósito está dirigido hacia ellos.

  Lo mismo ocurre con la relación expresada entre el MCUN y los artefactos GN y RN. Se sabe que cada uno de estos artefactos evolucionará paulatinamente con la construcción del modelo general.
- Dado que los términos definidos en el GN poseen una descripción textual única que debe ser respetada por todo elemento del modelado de negocio, podría indicarse en el diagrama de clases con una relación de dependencia desde cada

uno de los artefactos al GN. No figuran estas relaciones en el diagrama, para no complicar su visualización, y se indica de manera textual en la regla adicional MN.13 expresada en la sección 4.3.1.

- Las reglas de negocio que conforman el artefacto RN existen en cualquier contexto bajo estudio, y por lo tanto se visualiza y especifica que el artefacto RN debe existir siempre que se construye un MCUN.

Este análisis aporta una importante definición a la construcción de una solución, en la que el desarrollador puede determinar qué artefactos son indispensables u obligatorios definir cualquiera sea el dominio a representar con el modelo del negocio. Luego, dependiendo del contexto del problema a resolver y de la envergadura del negocio a modelar, existen documentos que son suplementarios al MCUN, que pueden ser incluidos al modelo general y son expresados en el diagrama con las clases DAN, ESN, VN y VFO.

- La relación entre los documentos DAN, ESN, VN y VFO y el MCUN, se dibuja con una asociación dirigida a cada documento con multiplicidad 0..1, y con multiplicidad 1 en el extremo que asocia al MCUN.

  Esta representación pretende expresar que si bien estos documentos no siempre son necesarios construirlos, cuando existan deberán estar necesariamente relacionados con el MCUN y este artefacto deberá mirar las decisiones documentadas en tales documentos, porque en muchos casos contendrán implícitamente información que puede afectar al flujo de sucesos de un proceso de negocio o simplemente debe ser trasladada a las funcionalidades del sistema.
- A partir de las relaciones expresadas entre el MCUN y sus componentes CUN y AN, el desarrollador encuentra que para construir el MCUN deberá tener en cuenta que por lo menos debe definir una instancia de CUN y una de AN, que no podrá existir una instancia de AN que no esté asociado a por lo menos una instancia de CUN y viceversa, y que en el caso de que exista solo una instancia de CUN esta deberá ser no abstracta o ser un caso de uso primario.

  Se considera caso de uso abstracto a aquel que nunca es utilizado por si mismo por un actor, sino que solo se utiliza desde otros casos de uso, comúnmente con relaciones de <<include>>. En este modelo genérico se denominan CUNAbstracto y CUNNoAbstracto, respectivamente, donde un los CUNNoAbstracto puede tomar servicios de CUNAbstracto, y cualquier CUNAbstracto sirve para prestar servicios a por lo menos un CUNAbstracto.
- De acuerdo a la definición dada por el RUP, cuando el análisis del negocio es bien entendido por todo el equipo del proyecto, el MAN puede ser excluido. Si ésta es la

decisión, en el modelo genérico es claro ver que, eliminando el MAN se eliminan todas sus componentes asociadas, y esto ocurre por la propia definición de la relación de agregación compuesta o composición, donde existe una fuerte relación de pertenencia y vidas coincidentes de la parte con el todo, en este caso entre el MAN y los elementos del modelo RCUN, EN, WN, EvN y SN.

- En algún caso, el MAN puede construirse de manera completa o puede alcanzar con definir solo el modelo de dominio, formado por las entidades del negocio (EN). El modelo de clases propuesto especifica de manera precisa que si el MAN es realizado, deben ser definidas por lo menos una o más EN, que usa al MCUN y que tanto el documento RN como el GN podrán ser utilizados, revisados y hasta actualizados cuando el MAN avance en su desarrollo.
- El MAN está relacionado con el MCUN dado que su propósito es describir cómo los CUN se desarrollan o ejecutan. Por lo tanto en el diagrama, se visualiza una relación semántica entre ambos modelos representada con una dependencia, indicando que el MAN usa la definición del MCUN, en la cual un cambio en el MCUN, elemento independiente, puede afectar la semántica del MAN, el elemento dependiente.
- Si bien todos los documentos del modelo poseen un propósito y una importancia en particular, el documento que conserva las reglas del negocio, RN es un artefacto de vital importancia dado que cada regla define una restricción o invariante que el negocio debe satisfacer y debe mantenerse durante todo el proyecto. Las restricciones pueden ser de comportamiento o estructurales, y afectan tanto a la secuencia de acciones que definen un CUN, como la pre y poscondiciones que deben satisfacerse para la RCUN.

Cuando se construye el modelo de dominio, generalmente representado con un diagrama de clases, se ven reflejadas reglas estructurales como relaciones y/o cardinalidad entre EN. Además, las reglas que componen el artefacto RN están también involucradas con el GN, dado que el mantenimiento de dichas políticas y condiciones durante todo el proyecto exigen una revisión permanente de los términos definidos en el GN.

- Se ven muchas otras relaciones entre los *elementos de modelos*. En la clase EN puede verse una relación de asociación a sí misma, y la multiplicidad expuesta indica que una instancia de EN puede estar relacionada con cero o más instancias de EN. Con el modelado visual, no es fácil indicar que una instancia de una determinada EN puede estar asociada a otra instancia de la misma entidad. También, una relación de asociación a sí misma puede verse en la clase CUN, solo que en este caso la relación no podrá darse entre instancias del mismo CUN,

por la propia definición de las relaciones posibles entre casos de uso (include, extend y generalización). Estas ambigüedades presentadas por el modelado visual con UML, y conocida por todos, se resuelven agregando reglas adicionales como puede verse en la sección siguiente.

Otras lecturas podrían hacerse del diagrama de clases de la figura 17. En la siguiente sección se exponen un conjunto de reglas que se obtienen del modelo genérico representado en el diagrama de clases y otras que lo complementan.

Entre las reglas listadas en las tablas 4.2., 4.3. y 4.4., algunas pueden ser leídas directamente del diagrama de clases propuesto, pero son igualmente expuestas por considerar que esta lista de reglas resume el análisis expresado en esta sección y la lectura realizada del diagrama que puede servir de mucho al desarrollador para su interpretación y aplicación.

4.3. Definición de Reglas

Como se dijo en la introducción de este capítulo, se definieron reglas o condiciones que complementan al modelo genérico, teniendo en cuenta que UML no permite, en algunos casos, expresar con claridad y sin ambigüedad, ciertas condiciones que el modelo genérico impone para la construcción de un modelo de negocio concreto, o la validación de que un modelo de negocio previamente construido cumple estas reglas y por lo tanto es correcto.

En esta sección las reglas están definidas en lenguaje natural. En el siguiente capítulo, se expone y justifica una formalización del modelo genérico, tanto del diagrama de clases como de aquellas reglas que complementan su definición, y no pueden ser deducidas o leídas del diagrama de clases. La formalización se realiza en el lenguaje de especificación formal Object-Z.

Las reglas definidas están agrupadas en reglas generales para el Modelo de Negocio, reglas asociadas a los elementos del Modelo de Casos de Uso del Negocio, y por último reglas que se corresponden con elementos del Modelo de Análisis del Negocio. Cada regla es identificada con la sigla MN, MCUN y MAN de acuerdo al grupo al que pertenece y seguida de un número correlativo.

El listado de reglas que se detalla a continuación, corresponde a reglas que imponen principalmente relaciones y condiciones entre los artefactos, sin considerar todas las condiciones a tener en cuenta en la construcción interna de cada artefacto propiamente dicho.

4.3.1. Reglas Generales para el Modelo de Negocio (MN)

Se presentan las reglas generales definidas para el Modelo de Negocio, teniendo en cuenta que MN es el modelo genérico y mn es una instancia del mismo. La Tabla 4.1. de la sección anterior muestra el significado de cada sigla y la representación de una sigla en particular corresponde a una instancia del artefacto que la misma representa. La Tabla 4.2. muestra las reglas generales definidas de forma genérica para el modelo de negocio. Desde MN.1 hasta MN.5 inclusive corresponden a declaraciones de la sintaxis del modelo genérico. A partir4 de MN.6 son reglas que deben ser consideradas en la construcción de un modelo de negocio específico.

Tabla 4.2. Reglas Generales para el Modelo de Negocio.

MN.1) MN: {artefacto} Indica que el Modelo de Negocio está compuesto por un conjunto de artefactos.
MN.2) artefacto = (model | modelElement | document) Cada artefacto corresponde a un tipo específico.
MN.3) model = (MCUN | MAN). Un model es un MAN o un MCUN.
MN.4) modelElement = (CUN | AN | SN | EN | WN | RCUN | EvN)
MN.5) document = (GN | RN | ON | ESN | DAN | VN | VFO)
MN.6) Cada artefacto de tipo document tiene una única instancia para un MN específico.
MN.7) Todo MN específico debe contener un GN, un ON, un RN y un MCUN.
MN.8) Todo artefacto identificado con <<document>>, está asociado al MCUN y es usado por este.
MN.9) La creación y actualización de los elementos del MAN depende de los elementos del MCUN.
MN.10) Cuando se crea una instancia del MCUN, deben existir las instancias de GN, RN y ON asociadas.
MN.11) Cuando se crea una instancia del MAN, deben existir las instancias del GN y RN asociados a él.
MN.12) Cada regla de negocio definida en RN se traslada o asocia a algún elemento del MCUN o del MAN.
MN.13) Todos los artefactos del MN respetan la definición única dada por los términos descriptos en el GN.
MN.14) Cada término del GN debe estar incluido en la descripción de por lo menos un CUN.

MN.15) Cada término definido en el GN que representa una entidad, EN, se corresponde con una clase del modelo de dominio.

4.3.2. Reglas definidas para el Modelo de Casos de Uso de Negocio (MCUN) y sus componentes

En la Tabla 4.3. se presentan las reglas definidas para el Modelo de Casos de Uso de Negocio, los artefactos que lo componen y otros artefactos que participan y están representados en el diagrama de clases.

Tabla 4.3. Reglas definidas para los Elementos del Modelo de Casos de Uso de Negocio.

MCUN.1)	MCUN: {modelElement}
MCUN.2)	modelElement= (CUN | AN)
MCUN.3)	Si existe una instancia de MCUN deben existir por lo menos una instancia de un CUN y un AN.
MCUN.4)	Si se elimina una instancia del MCUN desaparecen todos los cun:CUN y an:AN asociados a él.
MCUN.5)	Un cun:CUN no abstracto está siempre relacionado a por lo menos un an:AN.
MCUN.6)	Un cun:CUN abstracto no se relaciona con ningún AN.
MCUN.7)	Un cun:CUN abstracto debe poseer relación con al menos un cun:CUN no abstracto.
MCUN.8)	Un cun:CUN se relaciona con otro cun:CUN a través de relaciones include, extend o generalización.
MCUN.9)	Un cun:CUN no puede estar relacionado a si mismo.
MCUN.10)	Un an:AN está siempre asociado con al menos un cun:CUN no abstracto.
MCUN.11)	Un an:AN puede relacionarse con otro an:AN a través de relación de generalización.
MCUN.12)	Un an:AN no puede estar relacionado a si mismo.

4.3.3. Reglas definidas para el Modelo de Análisis del Negocio (MAN) y sus componentes

En la Tabla 4.4. se presentan las reglas definidas para los elementos del Modelo de Análisis de Negocio y su relación con otros artefactos.

Tabla 4.4. Reglas definidas para los elementos del Modelo de Análisis del Negocio.

MAN.1)	MAN: {modelElement}
MAN.2)	modelElement= (RCUN | SN | EN | WN | EvN).
MAN.3)	Si se elimina el MAN desaparecen todas las instancias de los elementos de EN, WN, SN, RCUN y EvN que lo componen.
MAN.4)	Si existe un MAN debe existir por lo menos un en:EN.
MAN.5)	Una en:EN representa información persistente manipulada por algún an:AN y/o wn:WN.
MAN.6)	Una en:EN puede relacionarse a si misma o a otras en:EN.
MAN.7)	El conjunto de EN y sus relaciones forman el Modelo de Dominio. Este modelo está incluido en el MAN.
MAN.8)	Una rcun:RCUN representa la interacción entre algun cun:CUN y an:AN.
MAN.9)	Una rcun:RCUN se corresponde con una o más interacciones producidas entre alguna wn:WN y en:EN.
MAN.10)	Cada instancia de WN, EN y EvN deben participar en al menos una rcun:RCUN.
MAN.11)	Una instancia de SN encapsula un conjunto de instancias de WN y de EN, y los EvN producidos para cumplir un propósito específico.
MAN.12)	Una instancia de un WN representa un rol desarrollado en una rcun:RCUN.
MAN.13)	Un wn:WN colabora con otros wn:WN.
MAN.14)	Cada wn:WN se asocia a por lo menos una en:EN para desarrollar sus responsabilidades.
MAN.15)	Cada en:EN debe documentarse en el GN.
MAN.16)	Un evn:EvN puede ser disparado y recibido en la interacción producida entre un wn:WN y una en:EN en una instancia de un RCUN.

4.4. Caso de estudio 1: Servicio de Información de una Tarjeta de Crédito

A continuación se desarrolla la instanciación del modelo con un caso de aplicación real para mostrar los alcances del método propuesto.

En este ejemplo se construyen y muestran las relaciones entre los artefactos: GN, ON, RN, MCUN y EN (MAN). La elección de estos artefactos responde a que, basado

en el modelo genérico y las reglas, el MCUN no puede ser construido de manera aislada, por el contrario necesita creados los artefactos GN, ON y RN. Además, para crear el MAN o una parte de él, se necesita el MCUN.

Con esta aplicación del modelo genérico se puede visualizar que se impone un orden en la creación de algunos artefactos, los cuales obviamente requerirán revisiones permanentes durante toda la construcción del modelo concreto.

Ejemplo: Se quiere construir el modelo de negocio del servicio de información que presta una hipotética empresa administradora de una Tarjeta de Crédito, para lo cual se dispone de la siguiente narrativa:

4.4.1. Descripción del Negocio

El servicio de información funciona a través del teléfono y para eso la empresa cuenta con un número que posee líneas rotativas. El promedio de llamadas es de 150 por día y la empresa presta el servicio de información durante 8 horas por día, en el horario de 9:00 a 17:00 horas. La franja horaria de mayor demanda es de 10:00 a 13:00 horas, donde se producen el 60% del promedio de llamadas. La empresa ha dispuesto que de 9:00 a 10:00 hs y de 13:00 a 17:00 hs estará un único operador atendiendo los llamados por una línea, mientras que en la franja horaria de mayor demanda atenderán tres operadores, intentando de esta manera satisfacer las necesidades de sus clientes.

Un llamado es descrito de la siguiente manera: un cliente realiza un llamado al número correspondiente al servicio, es atendido por un operador y lo consigna en una ficha de registración de llamados con los siguientes datos: apellido y nombre del cliente, número de documento, número de tarjeta, código de seguridad que figura al dorso de la tarjeta, una descripción y un código correspondiente al motivo del llamado. La empresa cuenta con una categorización de los motivos posibles de llamados, éstos son: extravío de tarjeta, compras en Internet, movimientos y saldo, vencimientos y pagos, promociones y asistencia al viajero.

La ficha de registración de llamados es abierta por el operador al iniciar su turno y la cierra al finalizarlo, luego esta ficha es archivada en un repositorio común de fichas de registración de llamados.

4.4.2. Creación del Modelo de Negocio en base al modelo genérico

Para crear el modelo de negocio del servicio de información de la tarjeta de crédito, se definen los artefactos GN, ON y RN en el orden mencionado. Luego, en base a estos artefactos se construye el MCUN y sus componentes CUN y AN, y por último se

modelan las EN identificadas y agrupadas con un modelo de dominio, correspondiente al MAN.

En la subsección 4.4.2.1 se describen los pasos a seguir para obtener un modelo de negocio específico basado en el modelo genérico propuesto, y en la subsección 4.4.2.2 se discute la aplicación del modelo genérico y el cumplimiento de las reglas.

4.4.2.1. Pasos a seguir para obtener la solución buscada

1) En primer lugar se analiza cuidadosamente el problema a resolver y se identifican los términos más importantes que definen este negocio. De acuerdo a la definición dada por el RUP [RUP] para la definición de los términos del glosario, se realiza una descripción textual de cada uno, la que deberá ser tenida en cuenta durante todo el proyecto en desarrollo. Estos términos son documentados en un artefacto denominado ***Glosario de Términos***, identificado en el modelo genérico como GN, y que serán refinados a medida que se construyan el resto de los artefactos.

   A continuación se muestra la instancia de GN para el ejemplo.

Artefacto: GN

- **Línea**: línea telefónica por la que el cliente se comunica con un operador para realizar una consulta.
- **Llamada**: acción que ejecuta el cliente para comunicarse con un operador y realizar una consulta.
- **Motivo**: es la causal de una consulta. Los motivos están categorizados en: Extravío de tarjeta, Compras en Internet, Movimientos y Saldo, Vencimientos y Pagos, Promociones y Asistencia al viajero.
- **Operador**: persona designada para atender las consultas de los clientes.
- **Cliente**: persona que posee una tarjeta de crédito de esta empresa y realiza consultas a través de la línea telefónica.
- **Ficha**: es una hoja compuesta por cinco columnas donde se registra el apellido y nombre del cliente, número de documento, número de tarjeta, código de seguridad, el código del motivo de la llamada y la descripción de la consulta.

2) El segundo paso, consiste en definir los ***Objetivos del Negocio***, aquellos que permiten hacer concretas y mensurables las estrategias del negocio. En este sencillo ejemplo sólo se define un objetivo de negocio y se documenta en el ON. En la documentación presentada por RUP [RUP], los objetivos de negocio se definen con un conjunto de propiedades mucho más extenso que solo el nombre y la descripción como se presenta aquí, el fundamento es que el modelo genérico tiene por objeto determinar los artefactos construir y su relación con los demás, y por lo tanto alcanza con esta definición del objetivo del negocio.

Artefacto: ON

Identificación del objetivo: Atender siempre una llamada.

Descripción: toda vez que un usuario realiza una llamada la empresa debe atenderla y satisfacer su inquietud, siempre que el motivo corresponda a una categoría preestablecida.

3) Una vez definidos los objetivos y los términos más importantes, el siguiente paso consiste en la definición de las políticas o ***Reglas del Negocio*** que rigen su funcionamiento y control. Siguiendo las recomendaciones dadas por RUP [RUP] para la definición de las reglas, se enumeran una a una usando el lenguaje natural o cualquier otro lenguaje semiformal o formal.

Artefacto: RN

1. Siempre debe haber por lo menos una línea habilitada y un operador disponible.
2. Un operador puede atender cualquiera de las líneas existentes en cualquiera de sus turnos.
3. Un cliente puede realizar todas las llamadas que desee para realizar sus consultas.
4. A una llamada se le asocia un único motivo.
5. Una llamada es atendida por una única línea.
6. Si el motivo de una llamada no está categorizado por la empresa, entonces los datos de esa llamada no deben ser registrados.

7. Si el motivo de la llamada está categorizado entonces debe registrarse dicha llamada en la ficha de registración abierta.
8. Una ficha de registración de llamadas debe ser abierta y cerrada por el mismo operador.

4) El siguiente paso es identificar y describir los ***Casos de Uso*** y ***Actores de Negocio***, que forman parte del ***Modelo de Casos de Uso del Negocio***. Con el fin de presentar un ejemplo muy sencillo, solo se define un caso de uso de negocio y dos actores que interactúan con él.

   Para el caso de uso definido *Realizar Consulta* se especifica una descripción informal, pre y pos condiciones, actores involucrados y el flujo de eventos. Los actores de negocio Operador y Cliente están definidos como términos en el GN. Además, todos los términos del GN son usados en el flujo de eventos y se destacan porque están subrayados. La figura 18 muestra un diagrama de casos de uso de negocio de UML que modela el caso de uso de negocio definido y los actores propuestos.

Artefacto: MCUN

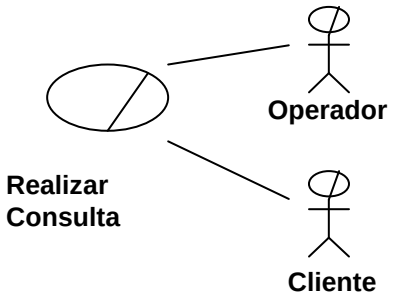


Figura 18: Diagrama de Casos de Uso de Negocio del Servicio de Información de la Tarjeta de Crédito

Descripción textual del artefacto CUN: Realizar Consulta

Un cliente realiza una llamada, un operador atiende y le pregunta los siguientes datos: apellido y nombre, número de documento, número de tarjeta, código de seguridad, descripción de la consulta y la categorización del motivo del llamado.

El operador consigna estos datos en su ficha de registración de llamados abierta, solo si el motivo del llamado corresponde a una categoría existente, caso contrario, los

datos del llamado no son almacenados en la ficha. Luego responde la consulta realizada por el cliente o no puede satisfacer su inquietud y corta la comunicación.

Caso de Uso de Negocio: *Realizar Consulta*
Actor de Negocio: Operador y Cliente
Pre-condición: hay una línea habilitada y un operador disponible dentro de la franja horaria de atención a los clientes. El cliente que llama posee una tarjeta de crédito de la empresa.
Acciones: 1. El cliente realiza una llamada. 2. El operador atiende la llamada y registra en una ficha los siguientes datos: Apellido y Nombre, Nº de documento, Nº de Tarjeta, Código de Seguridad, Motivo y Descripción de la consulta. 3. El operador responde la consulta. 4. El operador corta la comunicación.
Camino Alternativo: 2.1. Si el motivo de la llamada no corresponde a una categorización de la empresa no se registra la llamada y se corta la comunicación.
Pos-condición: el cliente realizó la consulta satisfactoriamente o la consulta no corresponde a un motivo válido de llamada.

5) El ***Modelo de Dominio*** es un subconjunto del ***Modelo de Análisis del Negocio*** (MAN) que facilita la comprensión del contexto del negocio, dado que define las entidades más importantes y la relación entre ellas. Es comúnmente representado con un diagrama de clases UML y está formado por las entidades del negocio, identificadas como EN en el modelo genérico.

Artefacto: EN

Modelo de Dominio del Servicio de Información de la Tarjeta de Crédito

Las EN que forman el modelo de Dominio son: Línea, Operador, Cliente, Motivo, Ficha, Llamada. El diagrama de clases de la figura 19 representa gráficamente el modelo de dominio construido.

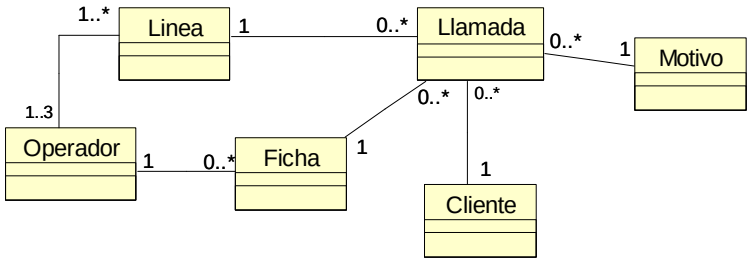


Figura 19: Diagrama de Clases del Modelo de Dominio del Servicio de Información de la Tarjeta de Crédito

4.4.2.2. Análisis de la aplicación del modelo genérico al caso de estudio

El caso de estudio desarrollado es sumamente sencillo con el objeto de facilitar la visualización del cumplimiento de las condiciones y reglas propuestas por el modelo genérico. El ejemplo presenta una instancia del modelo genérico propuesto en este trabajo mostrado en la figura 17 y también satisface las reglas definidas en la sección 4.3. previamente.

En primer lugar, el caso de estudio cumple con las reglas MN.7 y MN.10, las cuales imponen los artefactos que no deben faltan en todo modelo de negocio y en un orden de creación determinado. Observando el diagrama de clases (Figura 17) y específicamente atendiendo las relaciones entre el artefacto MCUN y los artefactos GN, ON y RN, puede verse que la multiplicidad indica que una instancia de MCUN posee relación con una instancia de ON, una de GN y una de RN. Y además, en la construcción de un modelo particular no podrá crearse el artefacto MCUN si antes no se ha creado una instancia de los artefactos mencionados.

Además, el ejemplo muestra una serie de pasos seguidos para obtener la solución, numerados del 1 al 5. El desarrollo se realiza en este orden para asegurar el cumplimiento de las reglas desde MN.7 a la MN.11, las cuales hacen referencia a la creación de las instancias de los artefactos y la dependencia entre los mismos, e indican cuales son los artefactos que deben estar en cualquier modelo de negocio particular.

Observando el desarrollo del ejemplo, se verifica el cumplimiento de las reglas desde MN.12 a MN.15. En detalle:

- MN.12 define que una regla de negocio perteneciente al artefacto RN que representa una pre o poscondición de una operación se traslada al flujo de eventos de algún elemento de CUN, y en el ejemplo puede tomarse en cuenta que la regla de negocio R1 (“Siempre debe haber por lo menos una línea habilitada y un operador disponible”) definida en el paso 3, se traslada a la

precondición del flujo de eventos que describe el caso de uso de negocio "Realizar Consulta".

- Respecto al cumplimiento de la regla MN.13 que indica que todos los términos poseen una definición única y es respetada durante todo el proyecto. Puede verse en la pre y poscondición del CU, en su descripción, en la multiplicidad entre las clases del modelo de dominio y en la definición de las reglas de negocio. Por ejemplo, la definición de los términos "llamada" y "cliente", son tenidas en cuenta tanto en la precondición del CUN "Realizar Consulta" como en su descripción.
- MN.14 define que todos los términos del GN, deben estar incluidos en el flujo de eventos de por lo menos un CUN. En el flujo de eventos del único CUN definido "Realizar Consulta" todos los términos del GN son usados y aparecen subrayados.
- El ejemplo cumple también con la regla MN.15 dado que todos los términos del GN que representan entidades del negocio, aparecen como clases del diagrama de clases que representa el modelo de dominio mostrado en el paso 5.

El ejemplo satisface las reglas MCUN.3, MCUN.5, MCUN.9 y MCUN.10 las que indican diversas relaciones entre el MCUN y sus componentes CUN y AN. Además, estas condiciones pueden extraerse directamente del diagrama de clases del modelo genérico, observando las relaciones entre estos artefactos y su multiplicidad. En detalle:

- MCUN.3 indica que un MCUN debe contener por menos un CUN y un AN. En la solución dada se identificaron un CUN y dos AN.
- MCUN.5 indica que un CUN no abstracto debe tener siempre por lo menos un AN relacionado. El CUN denominado "Realizar Consulta" es un caso de uso de negocio no abstracto[5] y por lo tanto debe interactuar con por lo menos un actor. En el ejemplo interactúa con los dos AN definidos "Cliente" y "Operador", y esto también implica el cumplimiento de la regla MCUN.10 la cual define que un AN debe estar siempre asociado a un CUN no abstracto.
- MCUN.9 indica que un CUN no puede estar relacionado a si mismo y no requiere de análisis para ver que se cumple.

[5] CUN abstracto es una funcionalidad usada por otros CUN, en general no abstractos, para completar su flujo de eventos, y generalmente están asociados a través de relaciones de <<incluye>>. Los CUN abstractos no poseen asociación con actores y en general son rehusados por varios CUN.

La única componente del MAN desarrollada en este ejemplo es EN, las entidades de negocio definidas en el paso 4. Estas entidades forman el modelo de dominio y se modelan con un diagrama de clases UML. Las condiciones asociadas a estos artefactos se cumplen para el ejemplo y las reglas van desde MAN.4 a MAN.7 y MAN.15. En detalle:

- MAN.4 impone que el modelo de análisis del negocio (MAN) puede estar definido parcialmente especificando solo las entidades más importantes del negocio EN. En el ejemplo se cumple esta condición.
- Para el caso de la regla MAN.5 se puede hacer un análisis de cuales son los datos de los cuales interesa mantener información guardada, y si las entidades definidas en el ejemplo cumplen dicho propósito. Lógicamente, se visualiza esta situación en el ejemplo desarrollado.
- MAN.6 indica que cualquier entidad del negocio puede relacionarse con otras entidades del mismo negocio que se está modelando, teniendo en cuenta que también puede presentar relaciones consigo misma. En el ejemplo desarrollado todas las entidades presentan relaciones estructurales con otras.
- La definición dada por la regla MAN.7 indica que el conjunto de entidades de EN y sus relaciones forman el Modelo de Dominio del negocio, y este modelo está incluido en el MAN.
- MAN.15 impone que cada entidad de negocio del EN debe documentarse en el GN y se cumple para el ejemplo dado.

En resumen, la instanciación del modelo genérico desarrollada en el ejemplo y el cumplimiento de las reglas presentadas en la sección 4.3 ilustra que la descripción en términos de clases y relaciones adoptada posibilita realizar de manera sencilla y sistemática la construcción y verificación de un modelo de negocio específico.

4.5. Caso de estudio 2: Control de la Caldera de Vapor

A continuación se presenta la descripción del problema conocido como “Control de la caldera de vapor” (Steam Boiler Control). Este problema se ha constituido en un problema clásico para el campo de la especificación formal de sistemas, ya que reúne en un problema relativamente sencillo de entender, características de un sistema crítico.

Básicamente, el sistema está compuesto de una caldera de vapor para una planta nuclear con cuatro bombas, cada una con su propio controlador de supervisión, un dispositivo para medir la cantidad de agua, otro para medir la cantidad de vapor que

sale de la caldera, y los correspondientes programas de operación y control de transmisión de mensajes. El modelo del sistema debe considerar un comportamiento tolerante a fallos, es decir, el sistema debe seguir funcionando aún con fallos en alguno de sus dispositivos.

A continuación se describe textualmente y detalladamente el problema del Control de la Caldera de Vapor, tal como aparece en su enunciado original [ABL96 (Cap. A)].

4.5.1. Descripción detallada del problema

El sistema está físicamente compuesto por los siguientes elementos:

- Una caldera de vapor.
- Un dispositivo capaz de medir la cantidad de agua en la caldera.
- Cuatro bombas que llevan agua a la caldera.
- Cuatro controladores (uno por bomba), que supervisan las bombas.
- Un dispositivo capaz de medir la cantidad de vapor que sale de la caldera.
- Una consola de operador.
- Un sistema de transmisión de mensajes.

La caldera de vapor

La caldera se caracteriza por los siguientes elementos:

- Una válvula para la evacuación de agua. Se usa para vaciar la caldera en su la fase inicial.
- Su capacidad total.
- Un límite inferior M1 de agua. Si se rebasa por debajo este límite más de 5 segundos la caldera puede estropearse si se continúa expulsando vapor y no se repone agua.
- Un límite superior M2 de agua. Por encima de este límite más de 5 segundos la caldera puede estropearse si las bombas continúan llevando agua a la caldera sin la posibilidad de evacuar vapor.
- Un límite mínimo normal N1 que debe mantenerse durante la operación normal de la caldera (N1 >M1).
- Un límite máximo N2 que no debe superarse durante la operación normal de la caldera (N2 <M2).
- La cantidad máxima de vapor W que puede salir de la caldera (en litros/seg).
- El máximo gradiente U1 de incremento de la cantidad de vapor (en (litros/seg)/seg).
- El máximo gradiente U2 de decremento de la cantidad de vapor (en (litros/seg)/seg).

Todas las cantidades de agua se miden en litros.

El dispositivo de medición del nivel de agua

Este dispositivo facilita la siguiente información:

- La cantidad q (en litros) de agua en la caldera de vapor.

Las bombas

Cada una de las bombas tiene las siguientes características:

- Su capacidad P (en litros/seg)
- Su estado de funcionamiento: on y off.
- Arrancando: desde que se activa la bomba necesita 5 segundos para comenzar a bombear agua a la caldera.
- Parando: de efecto inmediato.

El controlador de la bomba

El controlador de cada bomba ofrece la siguiente información:

- Si el agua circula por la bomba o no.

El dispositivo medidor de vapor

Este dispositivo ofrece la siguiente información

- Cantidad de vapor que sale de la caldera.

Funcionamiento general del programa

El programa se comunica con las unidades físicas del entorno a través de líneas dedicadas. Para una aproximación puede suponerse que el retardo de la transmisión es despreciable.

El programa de control es un ciclo en principio infinito, donde en cada iteración, que se lleva a cabo cada 5 sg., se realizan las siguientes acciones:

- Recepción de mensajes de los sensores externos.
- Análisis de la información recibida.
- Transmisión de las órdenes a los actuadores externos.

Modos de operación

El programa de control funciona en 5 modos de operación: inicialización, normal, degradado, rescate y parada de emergencia. A continuación se describe cada uno.

Modo de inicialización

Es el modo inicial de funcionamiento. El programa espera la recepción del mensaje STEAM_BOILER_WAITING proveniente de las unidades físicas. En cuanto llega el mensaje se llevan a cabo una serie de comprobaciones:

- Que la cantidad de vapor que sale de la caldera sea 0. Si esto no es así el programa entra en el *modo de parada de emergencia*.
- Que la cantidad de agua en la caldera sea mayor que N1. Si no es así, se activa la bomba para llenar la caldera.
- Que la cantidad de agua en la caldera sea menor que N2. Si no es así, se activa la válvula para vaciar la caldera.
- Si se detecta un fallo en el medidor del nivel del agua, se entra en el *modo de parada de emergencia*.

En cuanto el nivel del agua esté entre N1 y N2 el programa pasa a emitir la señal PROGRAM_READY a las unidades físicas hasta que recibe la señal PHYSICAL_UNITS_READY emitida por cada una de éstas. En cuanto llega esa señal, el programa pasa al *modo normal* de operación si todas las unidades funcionan correctamente o al *modo degradado* si alguna no lo hace. Un fallo en la transmisión hace que el programa pase al *modo parada de emergencia*.

Modo normal

Es el modo habitual de funcionamiento de la caldera. El programa de control trata de mantener el nivel del agua entre N1 y N2 con todas las unidades funcionando correctamente. En cuanto el nivel del agua se sale de este rango el programa lo corrige activando o desactivando la bomba. Si se reconoce un fallo de la unidad de medición se procede a entrar en el *modo de rescate*. El fallo de cualquier otra unidad conduce al *modo parada de emergencia*.

Si el nivel del agua se acerca peligrosamente a los límites M1 o M2, el programa entra en el *modo parada de emergencia*. La evaluación del peligro se hace en base al comportamiento máximo de las unidades físicas. Un fallo en la transmisión hace que el programa pase al *modo parada de emergencia*.

Modo degradado

En este modo el programa trata de mantener el nivel del agua independientemente de los fallos en algunas unidades. Se supone, sin embargo, que el medidor del nivel del agua funciona correctamente. El comportamiento es similar al modo anterior. En el momento que se repara la unidad dañada se vuelve al modo anterior. Al igual que antes, si el nivel del agua se acerca peligrosamente a los límites M1 o M2, el programa

entra en el *modo parada de emergencia*. Un fallo en la transmisión hace que el programa pase al *modo parada de emergencia.*

Modo de rescate

El programa trata de mantener el nivel de agua aunque falle la unidad de medición del nivel del agua. Éste se estima en función de los máximos parámetros dinámicos de los componentes de la caldera. Por simplicidad se supone que no hay expansión térmica (es decir, los litros suministrados por la bomba son los que entran en la misma), Se considera que el medidor del vapor y la bomba funcionan correctamente.

En cuanto se repara el medidor de agua se vuelve al *modo degradado*. Se irá al *modo parada de emergencia* si ocurre cualquier cosa de las siguientes:

- Falla el medidor del vapor.
- Falla la unidad que controla la bomba.
- El nivel del agua se acerca peligrosamente a uno de sus límites.
- Hay un fallo en la transmisión.

Modo parada de emergencia

Se llega a este modo cuando hay un fallo crucial para el funcionamiento del sistema. El programa se detiene, siendo el entorno físico responsable de tomar las acciones oportunas.

Mensajes emitidos por el programa

Los siguientes mensajes son enviados por el programa a las unidades del sistema:

- MODE(m): en cada ciclo se envía a todas las unidades el modo actual de operación.
- PROGRAM_READY: Se envía este mensaje durante la fase de inicialización en cada ciclo para indicar que el programa está listo, hasta que recibe el mensaje PHYSICAL_UNITS_READY de las unidades físicas.
- VALVE: Se envía durante la fase de inicialización para indicar que se abra/cierre la válvula de evacuación de agua.
- OPEN_PUMP(n): Se envía a las unidades físicas para abrir la bomba número n.
- CLOSE_PUMP(n): Se envía a las unidades físicas para cerrar la bomba número n.
- PUMP_FAILURE_DETECTION(n): se envía repetidamente hasta que se recibe confirmación para indicar que el programa ha detectado un fallo en la bomba n.

- PUMP_CONTROL_FAILURE_DETECTION(n): se envía repetidamente hasta que se recibe confirmación para indicar que el programa ha detectado un fallo en el controlador de la bomba n.
- LEVEL_FAILURE_DETECTION: se envía repetidamente hasta que se recibe confirmación para indicar que el programa ha detectado un fallo en el dispositivo medidor del nivel de agua.
- STEAM_FAILURE_DETECTION: se envía repetidamente hasta que se recibe confirmación para indicar que el programa ha detectado un fallo en el medidor de caudal de vapor de salida.
- PUMP_REPAIRED_ACK(n): se envía por el programa para contestar al mensaje recibido de bomba reparada.
- PUMP_CONTROL_REPAIRED_ACK(n): se envía por el programa para contestar al mensaje recibido de controlador de bomba reparado.
- LEVEL_REPAIRED_ACK(n): se envía por el programa para contestar al mensaje recibido de medidor de nivel de agua reparado.
- STEAM_REPAIRED_ACK(n): se envía por el programa para contestar al mensaje recibido de medidor de caudal de vapor de salida reparado.

Mensajes recibidos por el programa

Los siguientes mensajes son emitidos por las unidades físicas y recibidos por el programa:

- STOP: Cuando se recibe tres veces seguidas indica que el programa debe realizar una parada de emergencia.
- STEAM_BOILER_WAITING: se recibe durante la fase de inicialización y hace que el programa comience su funcionamiento normal.
- PHYSICAL_UNITS_READY: se recibe en la fase de inicialización como contestación al mensaje PROGRAM_READY enviado previamente.
- PUMP_STATE(n, b): indica que la bomba correspondiente está abierta o cerrada (open/close). Debe estar presente en cada transmisión.
- PUMP_CONTROL_STATE(n, b): indica si hay flujo de agua o no por la bomba. Debe estar presente en cada transmisión.
- LEVEL(v): Contiene el nivel del agua en la caldera. Debe estar presente en cada transmisión.
- STEAM(e): Contiene información proveniente del medidor de caudal de vapor de salida. Debe estar presente en cada transmisión.

- PUMP_REPAIRED(n): Indica que la correspondiente bomba ha sido reparada. Se envía repetidamente hasta que la unidad correspondiente recibe el mensaje de contestación del programa.
- PUMP_CONTROL_REPAIRED(n): Indica que el correspondiente controlador de bomba ha sido reparado. Se envía repetidamente hasta que la unidad correspondiente recibe el mensaje de contestación del programa.
- LEVEL_REPAIRED: Indica que el medidor de nivel de agua ha sido reparado. Se envía repetidamente hasta que la unidad correspondiente recibe el mensaje de contestación del programa.
- STEAM_REPAIRED: Indica que el medidor de caudal de vapor de salida ha sido reparado. Se envía repetidamente hasta que la unidad correspondiente recibe el mensaje de contestación del programa.
- PUMP_FAILURE_ACK (n): es la contestación de la unidad física (Bomba) correspondiente al mensaje de detección de fallo emitido por el programa.
- PUMP_CONTROL_FAILURE_ACK(n): es la contestación de la unidad física (Controlador de Bomba) correspondiente al mensaje de detección de fallo emitido por el programa.
- LEVEL_FAILURE_ACK: es la contestación de la unidad física (Dispositivo medidor de Agua) correspondiente al mensaje de detección de fallo emitido por el programa.
- STEAM_FAILURE_ACK: es la contestación de la unidad física (Dispositivo medidor de Vapor) correspondiente al mensaje de detección de fallo emitido por el programa.

4.5.2. Modelado del problema de 'Control de la Caldera de Vapor' usando el Modelo Genérico del Modelo de Negocio

El sistema se ha especificado en varias técnicas y lenguajes formales de especificación como LUSTRE, SIGNAL, NUT, FOCUS, VDM, LOTOS, Statecharts, Z y otros, que pueden consultarse en [ABL96].

A continuación se intenta mostrar la especificación del problema del *"control de la caldera de vapor"* instanciando el modelo genérico del modelo de negocio, propuesto en esta tesis.

La solución a este problema más complejo, se desarrolla de igual manera que para el caso de estudio 1 (sección 4.4.). Se desarrollan los artefactos: GN, ON, RN, MCUN y EN (MAN), que de acuerdo al modelo genérico y sus reglas, son los artefactos necesarios y obligatorios de construir cualquiera sea el problema.

De acuerdo a los pasos descriptos en la subsección 4.4.2.1 se obtiene una solución para el problema planteado a partir de la definición del modelo de negocio basado en el modelo genérico propuesto. Y a luego se discute la aplicación del modelo genérico para este problema y la solución obtenida.

La técnica propone un orden en la creación de los artefactos, los cuales obviamente requerirán revisiones permanentes durante todo el desarrollo del modelo concreto.

Se comienza por la definición de los artefactos GN, ON y RN en ese orden, y sobre la base de estos artefactos se construye el MCUN y sus componentes CUN y AN. Por último se modelan las EN identificadas y agrupadas en un modelo de dominio, correspondiente al MAN.

1) Artefacto: GN

- **Caldera de Vapor**: elemento físico que emite vapor para una planta nuclear. Está compuesta por cuatro bombas de agua con un controlador para cada una, un dispositivo medidor de la cantidad de agua, un dispositivo medidor de la cantidad de vapor que sale de la caldera, una válvula para vaciar la caldera, y programas de operación y control de transmisión de mensajes.
- **Dispositivo medidor de agua**: elemento físico que permite medir el nivel de agua que está en la caldera.
- **Dispositivo medidor de vapor**: elemento físico que permite medir la cantidad de vapor que sale de la caldera.
- **Bomba**: elemento físico que permite agregar agua a la caldera.
- **Controlador de Bomba**: cada bomba tiene su propio controlador, que indica si el agua circula o no por la bomba.
- **Programas de control**: el programa se comunica con las unidades físicas del entorno a través de líneas dedicadas, a través del envío y recepción de mensajes.
- **Unidades físicas**: implica cualquier elemento físico que forma parte de la caldera (Bomba, Controlador de Bomba, Medidor de Agua, Medidor de Vapor)

2) Artefacto: ON

Identificación del objetivo: mantener siempre el nivel de agua aceptable en la caldera.

Descripción: un correcto control de la caldera implica que la misma mantenga su funcionamiento siempre que tenga un nivel de agua aceptable, caso contrario se deberán iniciar las acciones correctivas necesarias para que esto suceda.

3) Artefacto: RN

1. El programa de control debe ejecutarse cada 5 segundos automáticamente.
2. Suponer que el retardo de la transmisión de mensajes es despreciable.
3. Suponer que en cada ejecución, la emisión o recepción de todos los mensajes se hace de manera simultánea.
4. El sistema se basa en un comportamiento tolerante a fallos, es decir, en muchos casos debe seguir funcionando aún con fallos en alguno de sus dispositivos.
5. Detección de fallos de las unidades físicas

   Las condiciones de fallos consideradas por el programa de control para decidir si una unidad física está estropeada o no, son:

- Para la **Bomba**, las condiciones de error son:

  5.1. El programa ha enviado el mensaje para conectar o desconectar la bomba y en el siguiente ciclo no se recibe la confirmación de la operación.

  5.2. El programa detecta un cambio espontáneo de estado de la bomba.

- Para el **Controlador de la bomba**, hay dos posibles errores:

  5.3. El programa ha enviado un mensaje para conectar o desconectar la bomba, y en el 2º ciclo posterior no recibe la confirmación de que el agua circula o no, a pesar de tener constancia de que la bomba funciona.

  5.4. El programa detecta un cambio espontáneo de estado.

- Para la **Unidad de medida del nivel de agua**, los dos posibles errores son:

  5.5. El programa detecta que la unidad devuelve un valor imposible (menor que 0 ó mayor que C)

  5.6. Se detecta que la unidad devuelve un valor incompatible con la dinámica del sistema.

- Para la **Unidad de medida del caudal de vapor de salida**, los dos posibles errores son:

  5.7. El programa detecta que la unidad devuelve un valor imposible (menor que 0 ó mayor que W).

5.8. Se detecta que la unidad devuelve un valor incompatible con la dinámica del sistema.

- Para el **Sistema de transmisión de mensajes**, se suponen dos posibles errores:

5.9. Llegada de un mensaje aberrante.

5.10. Ausencia de un mensaje indispensable.

4) Artefacto: MCUN

Para el modelado de este artefacto se utilizan tres diagramas de UML. El diagrama de casos de uso de negocio (Figura 20), para representar los procesos más importantes identificados en este problema, un diagrama de actividades (Figura 21) para modelar el flujo de control de tareas que se llevan a cabo cuando se inicializa el funcionamiento de la caldera, y un diagrama de estados (Figura 22) para especificar y visualizar los posibles estados en los que permanece el programa de control para mantener el nivel de agua de la caldera.

Se utiliza el lenguaje natural para expresar tanto los eventos, como las condiciones de guarda y las acciones de las transiciones.

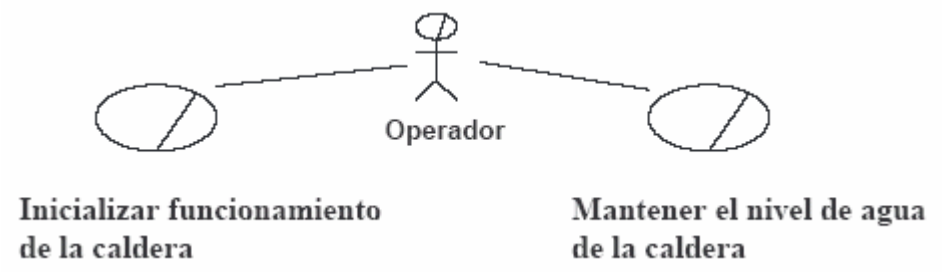


Figura 20: Diagrama de Casos de Uso de Negocio del Control de la Caldera de Vapor

Descripción textual del artefacto CUN: Inicializar funcionamiento de la caldera.

En el ***modo inicial*** de funcionamiento, el programa espera la recepción del mensaje STEAM_BOILER_WAITING proveniente de las unidades físicas. En cuanto llega el mensaje se llevan a cabo una serie de comprobaciones:

- Que la cantidad de vapor que sale de la caldera sea 0. Si esto no es así el programa entra en el ***modo de parada de emergencia***.
- Que la cantidad de agua en la caldera sea mayor que N1. Si no es así, se activa la bomba para llenar la caldera.
- Que la cantidad de agua en la caldera sea menor que N2. Si no es así, se activa la válvula para vaciar la caldera.

- Si se detecta un fallo en el medidor del nivel del agua, se entra en el ***modo de parada de emergencia***.

En cuanto el nivel del agua esté entre N1 y N2 el programa pasa a emitir una señal a las unidades físicas y cuando éstas le responden satisfactoriamente, el programa pasa al ***modo normal***. Si alguna de las unidades no funciona correctamente pasa al ***modo degradado***. Si falla alguna transmisión el programa pasa al ***modo parada de emergencia***.

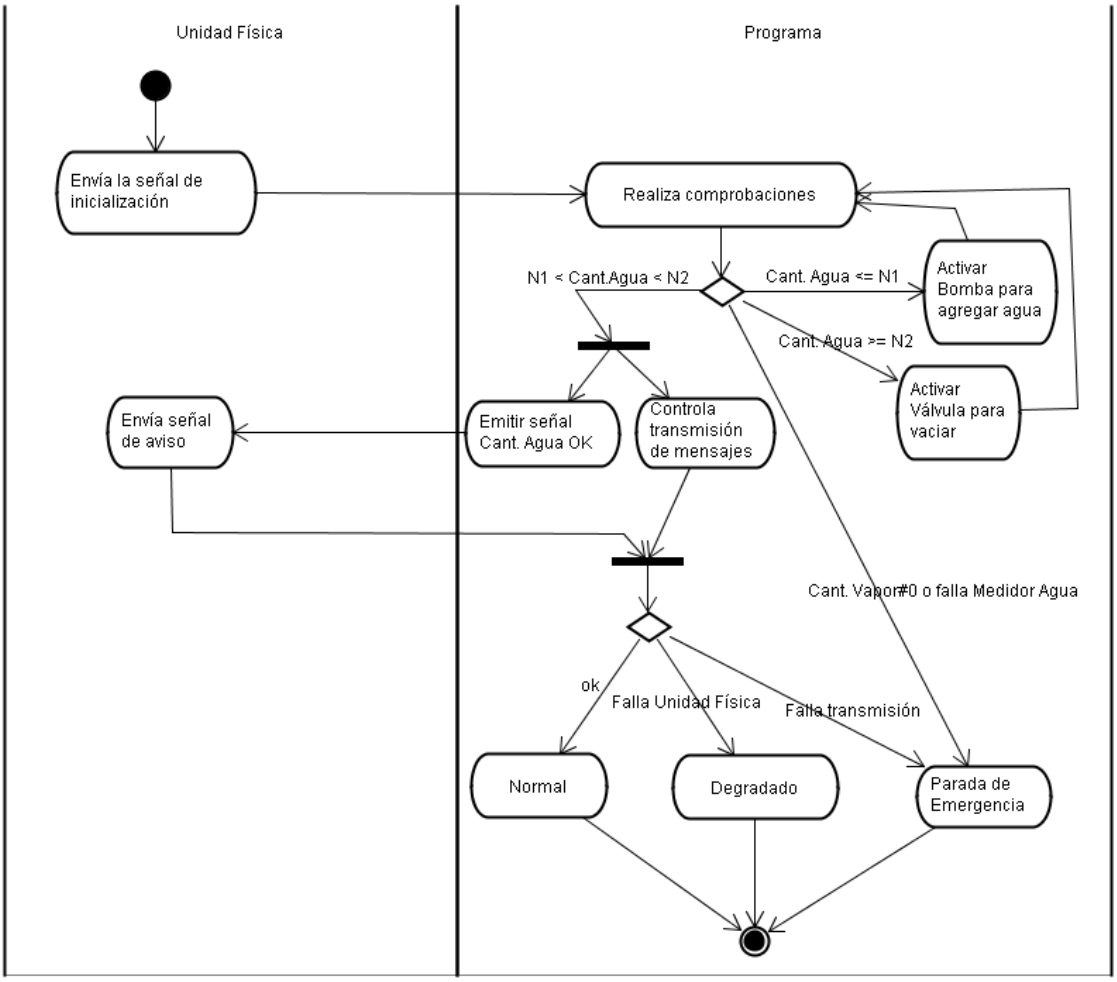


Figura 21: Diagrama de Actividades del CUN: Inicializar funcionamiento de la caldera.

Descripción textual del artefacto CUN: Mantener el nivel de agua de la caldera.

Estando en un ***modo normal***, el programa de control trata de mantener el nivel del agua entre N1 y N2 con todas las unidades funcionando correctamente.

Si el nivel del agua es menor o igual que N1 el programa lo corrige activando una bomba. En cambio, si el nivel del agua es mayor o igual que N2 el programa activa la válvula para vaciar la caldera.

Si el dispositivo medidor del nivel del agua funciona correctamente, pero falla alguna otra unidad, el programa entra en ***modo degradado***, y sale de este modo cuando la unidad fallada es reparada.

Si el dispositivo medidor del nivel del agua falla, pero el medidor del vapor y la bomba funcionan correctamente, el programa trata de mantener el nivel de agua hasta que éste es reparado y se queda en ***modo de rescate***. En cuanto se repara el medidor de agua se vuelve al ***modo degradado***.

El fallo de cualquier otra unidad, un fallo en el medidor de vapor, un fallo en la transmisión, o si el nivel de agua se acerca peligrosamente a M1 o M2, el programa se conduce al ***modo parada de emergencia***.

En el ***modo parada de emergencia***, el programa se detiene, siendo el entorno físico responsable de tomar las acciones oportunas.

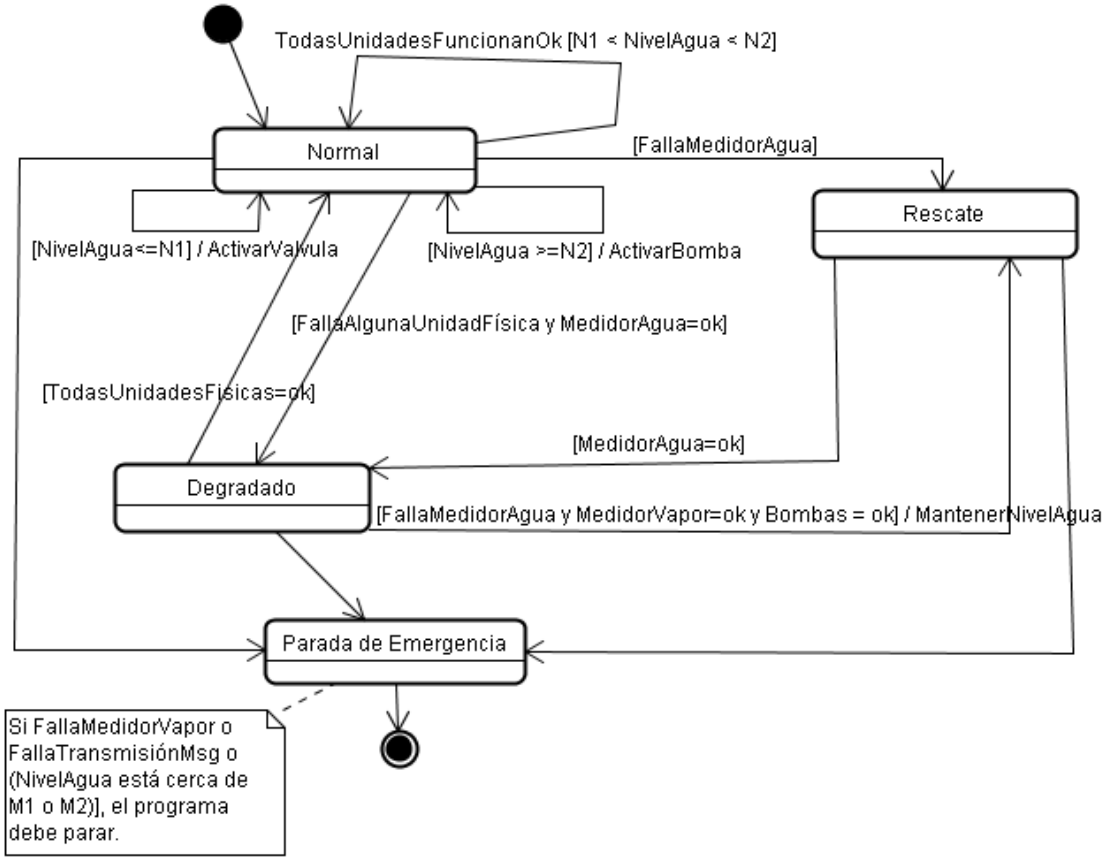


Figura 22: Diagrama de Estados del CUN: Mantener el nivel de agua de la caldera.

5) Artefacto: EN

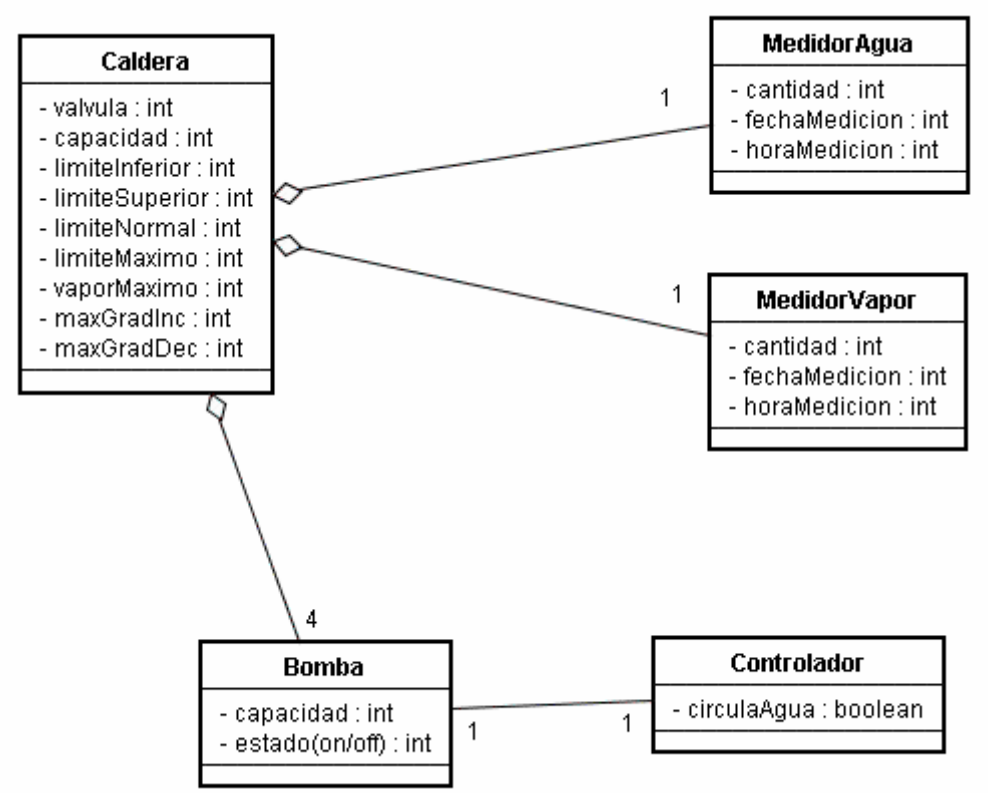


Figura 23: Diagrama de Clases del Modelo de Dominio del Control de la Caldera de Vapor

4.5.2.1. Análisis de la aplicación del modelo genérico al caso de estudio 2

Como ya se indicó, el principal objetivo del modelado del negocio es adquirir un conocimiento suficiente, necesario y significativo de la estructura y la dinámica del contexto bajo estudio. Con la aplicación del modelo genérico sobre este problema, se consigue analizarlo y obtener el conocimiento suficiente que se requiere de la estructura y el funcionamiento de la Caldera de Vapor, como así también de los elementos a considerar para el control de la misma.

Es importante tener en cuenta que si bien este problema se plantea como complejo, al aplicar el modelo genérico para obtener el modelo de negocio se ve que el problema no muestra mayor complejidad que el planteado en el caso de estudio 1. Esto tiene que ver con el objetivo perseguido por el modelado de negocio, y que se encuentra en una etapa muy temprana del desarrollo de un proyecto de software. Sin embargo, el desarrollador que utilice esta técnica, podría querer obtener un mayor detalle de la especificación de cada una de las actividades descriptas en la figura 22, o de los estados y transiciones modelados en la figura 23, debería desarrollar otros artefactos como la realización de los casos de uso de negocio (RCUN) que le permita describir cómo se ejecutarán dichas actividades o acciones dentro de la organización, desde

una perspectiva interna, ya que un caso de uso de negocio solo describe desde una perspectiva externa.

Todas las reglas planteadas por el modelo genérico son tenidas en cuenta para obtener esta solución, y por lo tanto se cumplen. Tomando el modelo genérico representado en el diagrama de clases de la figura 17, y las reglas adicionales definidas en la sección 4.3, un análisis muy similar al realizado para el caso de estudio 1 (4.4.2.2) podría ser realizado.

La aplicación del modelo genérico del modelo de negocio sobre este caso de estudio, permite arribar a una conclusión más general para cualquier caso real al que se aplique. Esta técnica brinda la posibilidad de realizar de manera sencilla y sistemática la construcción y verificación de un modelo de negocio específico, logrando el conocimiento anhelado y pretendido del problema bajo estudio. Esta técnica permite organizar los artefactos a través de relaciones y reglas y ayuda a los desarrolladores de software a identificar y definir los artefactos de un modelo de negocio particular, reduciendo problemas de ambigüedad en la definición y principalmente reduciendo el tiempo insumido por el desarrollador para analizar y priorizar los artefactos a construir para cada caso.

CAPITULO 5: Formalización del Modelo Genérico

5.1. Introducción

Una práctica común entre los ingenieros de software, es utilizar algún tipo de diagramas para modelar gráficamente sus sistemas. UML [BRJ99] [OMG] es una de las herramientas más poderosas y utilizadas para modelar gráficamente sistemas orientados a objetos. Provee un mecanismo de modelado gráfico muy útil para visualizar, especificar, construir y documentar los artefactos de un sistema, y permite que usuarios y desarrolladores se entiendan más fácilmente a través de un lenguaje común, fácil de entender y aplicar. Sin embargo, este lenguaje de modelado gráfico no posee una semántica precisa, y por lo tanto, se presentan ambigüedades que originan problemas de interpretaciones erróneas o diferentes.

Las especificaciones formales se expresan en notación matemática con una sintaxis y semántica definida en forma precisa. La especificación formal de sistemas permite complementar las técnicas de especificación informal.

Existen numerosos trabajos que presentan definiciones formales de la semántica de UML, a través de su traducción a lenguajes formales, tal como Object-Z [Smith95, Smith00]. Object-Z es un lenguaje de especificación formal de sistemas orientados a objetos, que surge como una extensión del lenguaje formal Z [Spivey92].

A continuación se muestra una breve reseña de algunos de los trabajos citados en esta área de interés. Kim y Carrington [KC99] [KC00] proveen las bases formales para la estructura sintáctica y semántica de los constructores de modelado de UML y definen un mecanismo de razonamiento acerca de estos modelos. Para la descripción formal de constructores UML, utilizan clases en Object-Z. Cualquier verificación de modelos UML puede hacerse con sus correspondientes especificaciones Object-Z, usando técnicas de razonamiento provistas por éstas. En el 2001 [KC01], estos mismos autores presentaron un metamodelo basado en la transformación entre UML y Object-Z, y en año 2004 [KC04] introducen la importancia de la definición de restricciones de consistencia entre los modelos UML, y utilizan Object-Z con el fin de demostrar cómo el metamodelo puede ser extendido a restricciones de consistencia para modelos UML. También P.Moura y otros [MBM03], proponen un uso práctico de los métodos formales donde fragmentos del lenguaje de especificación Object-Z son embebidos en diagramas de clases UML, también llamado integración entre UML y métodos formales. Tanto en el trabajo de Roe, y otros [RBR03] como en Becker y

Pons [BP03]; se desprende el mismo propósito, traducción de diagramas de clases UML complementados con expresiones OCL a expresiones Object-Z, con el fin de proveer una formalización de los modelos gráficos expresados mediante UML/OCL que permita aplicar técnicas clásicas de verificación y prueba de teoremas sobre los modelos. Ambos trabajos utilizan una notación muy similar para aplicar la traducción del lenguaje gráfico al lenguaje formal.

En este capítulo se propone la formalización de la semántica del modelo genérico del modelo de negocio, representado gráficamente con el diagrama de clases de UML (Capítulo 4) al lenguaje Object-Z.

Como ya se expresó anteriormente, si bien es valiosísima las ventajas que aportan los Diagramas de Clases de UML para el modelado de sistemas, también se sabe que no permiten precisar suficientemente el conjunto completo de aspectos a tener en cuenta en una especificación. En este sentido, se decide traducir el modelo genérico del modelo de negocio a un lenguaje de especificación formal, con el fin de garantizar que están correctamente modelados los modelos de negocio de dominios particulares que son construidos a partir de la especificación genérica presentada en esta tesis.

Se sabe que aún cuando la aplicación de métodos formales no garantiza la corrección a priori de un sistema, facilita considerablemente el análisis de las propiedades del sistema, mostrando posibles inconsistencias, ambigüedades o incompletitudes.

La selección del lenguaje de especificación Object-Z se funda principalmente en que este proporciona un formalismo y una semántica uniforme para la representación de modelos diagramáticos, tales como los diagramas de clase UML. El lenguaje preserva la mayoría de las características de las estructuras orientadas a objetos de los modelos semi formales de UML. Además, los modelos formales obtenidos con Object-Z parecen ser más accesibles a los ingenieros de Software, que otros lenguajes de especificación formal.

5.2. De UML a Object-Z

Object-Z [Smith95] es una extensión orientada a objetos de Z [Spivey92]. Es una extensión porque tanto la sintaxis como la semántica de Z son también una parte de Object-Z, es decir, una especificación Z es también una especificación Object-Z.

Es un lenguaje de especificación formal basado en estados, en el cual los estados de un sistema, el estado inicial y las operaciones son modeladas por esquemas que comprenden un conjunto de declaraciones de variables asociadas a un predicado. Una clase en Object-Z encapsula un esquema de estado, y asociado a un esquema de

estado inicial, con todos los esquemas de operación que pueden cambiar sus variables. Object-Z facilita la especificación formal de sistemas orientados a objetos, no fuerza ningún estilo particular de la especificación, pero proporciona construcciones que ayudan a especificar sistemas en una manera particular. Las construcciones proporcionadas permiten generar especificaciones usando clases, objetos, herencia y polimorfismo.

Object-Z permite hacer declaraciones de referencias a los esquemas de clase, porque el valor almacenado es el valor del identificador del objeto. Esto remueve el problema de actualizar las referencias del objeto (como en Z). Además, incorpora la constante self que implica la identidad implícita de cada objeto.

Para la traducción del diagrama de clases UML que modela el modelo genérico del modelo de negocio (figura 17, capítulo 4) a expresiones Object-Z, se tienen en cuenta diferentes aproximaciones expuestas en [BP2003], [KC99], [KC00] y [RBR03], y se define aquí un mecanismo de traducción que considera las características más importantes de este modelo. Este mecanismo de traducción es usado para formalizar en Object-Z el modelo genérico del modelo de negocio.

5.2.1. Mecanismo de Traducción

- En primer lugar, MN es definido como un paquete que contiene todas las clases del diagrama.
- **Clases**: por cada clase del diagrama se crea un esquema de clase en Object-Z, con el mismo nombre de la clase y se le antepone el nombre del paquete MN. Ver el ejemplo de la figura 24, clases A y B expresadas en UML y formalizadas con su respectivo esquema de clase en Object-Z.
- Los **estereotipos** de las clases UML son definidos como variables públicas del esquema de clase MN. Para cada clase obtenida en Object-Z, se define un atributo que indica el tipo de estereotipo que representa. Para MN se definen tres tipos básicos, llamados *Model*, *Element* y *Document*, que permiten indicar a que estereotipo corresponde cada artefacto traducido.
- Una **relación de asociación** entre dos clases del diagrama se expresa como atributos en las correspondientes clases creadas en Object-Z.

  Por cada extremo o final navegable de la asociación, se incluye en la clase involucrada un atributo que representa la clase asociada en el otro fin de asociación, y su tipo depende de la multiplicidad. En Object-Z es legítimo referirse a una clase antes de que esta haya sido definida.

  - Para las relaciones con multiplicidad 0..1 o 1, el tipo del atributo es el de la clase opuesta,

- Para relaciones con multiplicidad 0..*, el tipo se representa como un conjunto de elementos del tipo de la clase opuesta. En Object-z, esta propiedad se denota con el símbolo Π (Power Set) y es definida como todos los subconjuntos de un conjunto dado. Por ejemplo: si *a* es el conjunto {*x*, *y*} entonces: Π *a* = { 0, {*x*}, {*y*}, {*x*, *y*} }.
- Por cada extremo navegable de la asociación, se agrega una restricción a cada esquema de clase Object-Z de las clases involucradas. Por ejemplo, para la asociación entre las clases A y B de la figura 24, la restricción se escribe como: $\mathcal{A}$ *x:bes* $\propto$ *self* $\in$ *x.a*, en el esquema de clase de B, y en el esquema de clase de A la restricción es $\mathcal{A}$ *x:a* $\propto$ *self* $\in$ *x.bes*.
- Las asociaciones de una clase a si misma pueden ser formalizadas en Object-Z usando definiciones recursivas, con la constante *self*, dos atributos del tipo de la clase definida y un predicado que especifica la relación. En la figura 24, la relación etiquetada como *relbab* es un ejemplo de este caso.

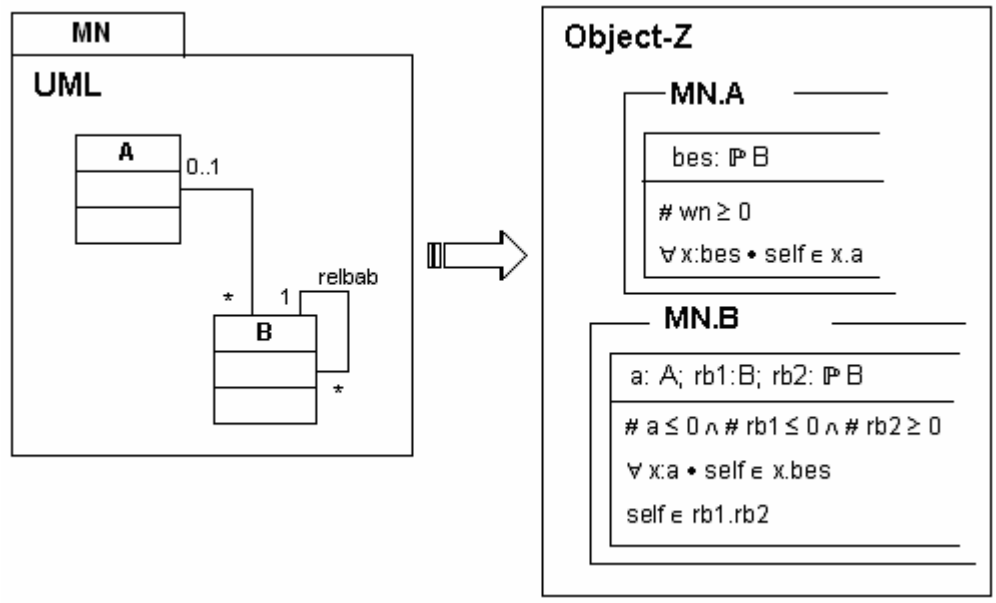


Figura 24: Relación de asociación de UML a Object-Z

- En general, la siguiente tabla indica la manera en que se escriben las restricciones de multiplicidad de acuerdo a los distintos tipos de multiplicidad dados para un atributo arbitrario x.

Multiplicidad	**En Object-Z**
0..1	# x ≤ 1
1	# x = 1
a..*	# x ≥ a para a ∈ N

- Si la asociación es una **composición** o **agregación** entre dos clases, en la clase que representa el *todo* se debe agregar el símbolo © a cada atributo que represente una *parte* de dicho *todo*. Además se agrega una restricción que indica que los elementos de la *parte* están contenidos o pertenecen al *todo*. En la Figura 25 se muestra un ejemplo.

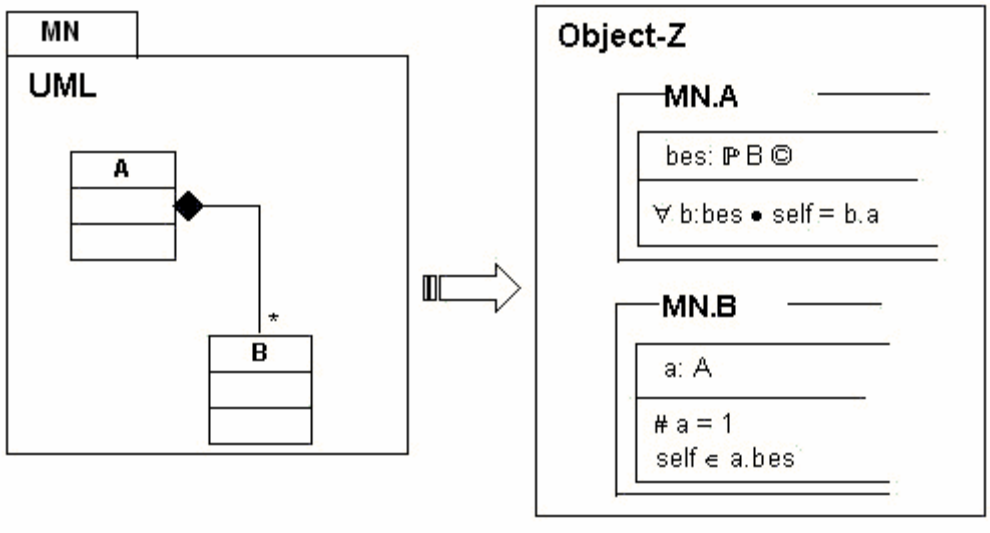


Figura 25: Relación de composición de UML a Object-Z

- Para traducir la **clase asociación** de UML a Object-Z, se define un esquema de clase con un atributo por cada clase asociada y una restricción que indica que solo existe una instancia de la clase asociación que pertenece a una instancia de las clases asociadas. (Figura 26).

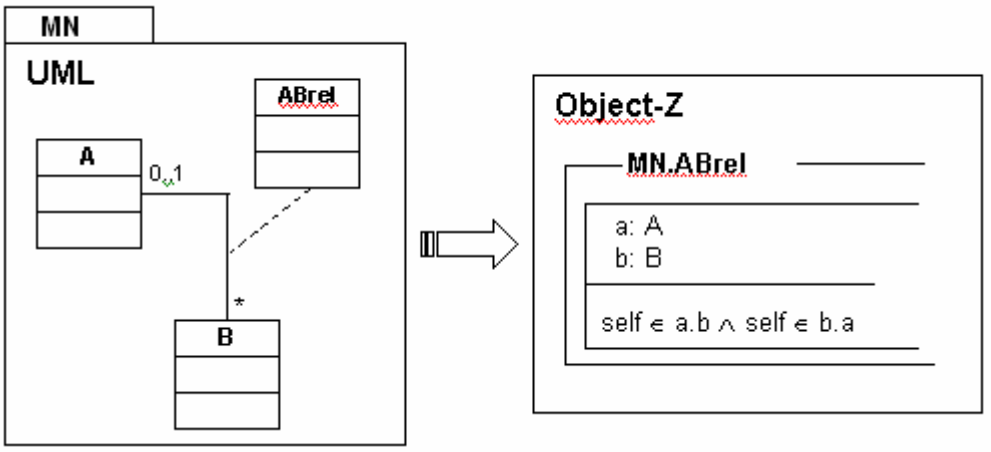


Figura 26: Clase asociación de UML a Object-Z

- Para el caso de la **generalización**, se agrega el nombre de la clase padre al comienzo del esquema de clase de la clase hijo (Figura 27).

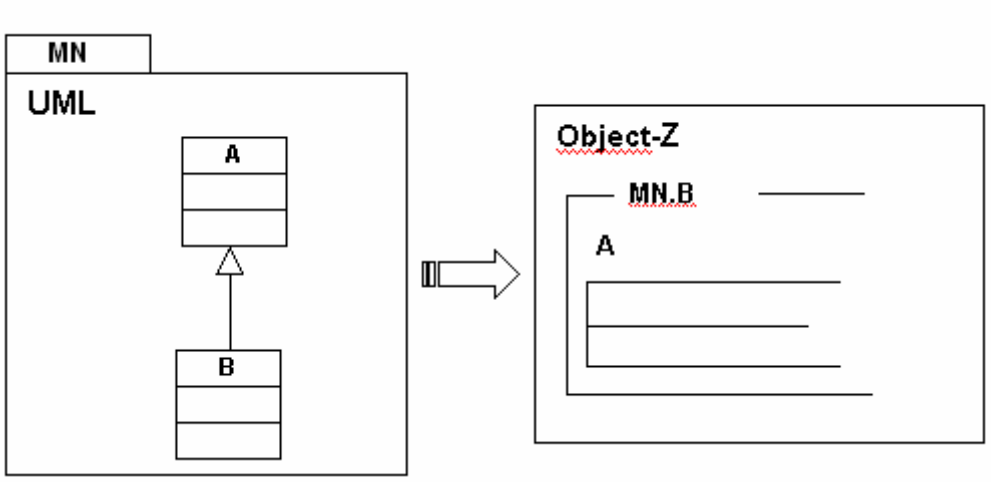


Figura 27: Relación de generalización de UML a Object-Z

El conjunto de restricciones o **reglas adicionales** al modelo genérico, expresadas en la sección 4.3. (Capítulo 4), escritas en lenguaje natural, son traducidas al lenguaje formal Object-Z en la sección 5.4..

5.3. Formalización del modelo genérico del modelo de negocio con Object-Z

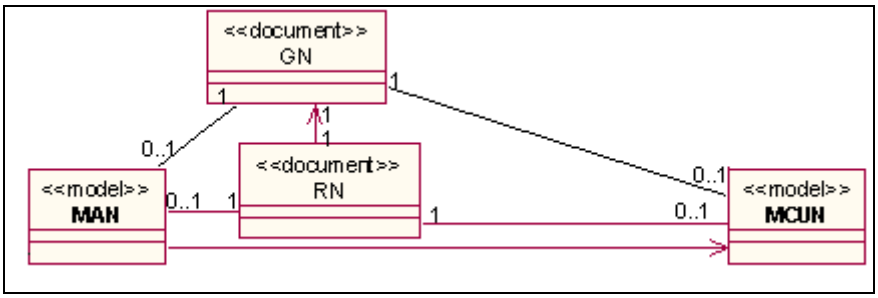
Tomando el modelo genérico del modelo de negocio, especificado con el diagrama de clases de la figura 17 (Capítulo 4), y siguiendo cada paso del mecanismo de traducción anteriormente expuesto, se obtienen las correspondientes expresiones en Object-Z. De acuerdo a este mecanismo, cada clase del diagrama es traducida a un esquema de clase de Object-Z.

Cada clase posee un atributo que indica el estereotipo de la clase. Los tipos de estos atributos son definidos abstractamente en Object-Z, al igual que en Z, como conjuntos dados.

Se parte con la formalización de la clase GN, que representa el glosario de términos del modelo de negocio, y en el diagrama de clases muestra una relación de asociación con navegabilidad en ambos extremos, con las clases MAN y MCUN. De acuerdo a la traducción definida en el mecanismo anteriormente descrito, el esquema de clase que representa a GN contendrá dos atributos, una restricción para indicar la multiplicidad y una restricción que indica la navegabilidad, ambas por cada atributo. También GN posee un atributo definido para el estereotipo de la clase, en este caso de tipo document.

En la figura 28 se transcribe la porción del diagrama de clases original, donde se visualiza el artefacto GN y sus relaciones, y su correspondiente traducción a Object-Z.

Además, MN es el paquete que contiene todos los artefactos modelados con esquemas de clases, GN es un elemento del MN, por lo tanto el identificador del esquema de clase contiene el nombre del paquete seguido del nombre del artefacto.



[Model, Element, Document]

Figura 28: Esquema de Clase del Glosario de Negocio - GN

El esquema de clase de la figura 29 traduce la clase RN del diagrama de clases original. La clase RN posee el estereotipo document, y relaciones de asociación con MCUN, MAN y GN. Respecto a la asociación de esta clase con GN, solo posee navegabilidad en el sentido de RN a GN y no a la inversa. Por lo tanto el esquema de clase de RN, además de los atributos mcun y man, tiene un atributo del tipo GN, tres restricciones que definen la multiplicidad y tres restricciones para la navegabilidad.

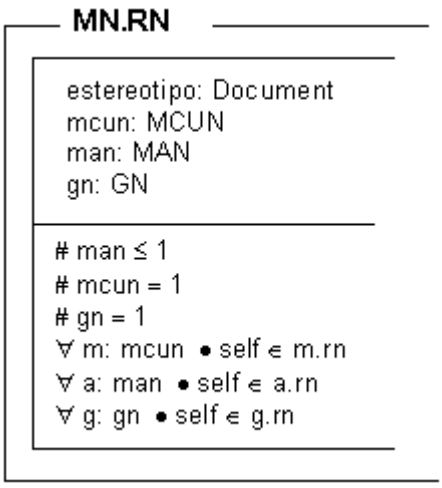
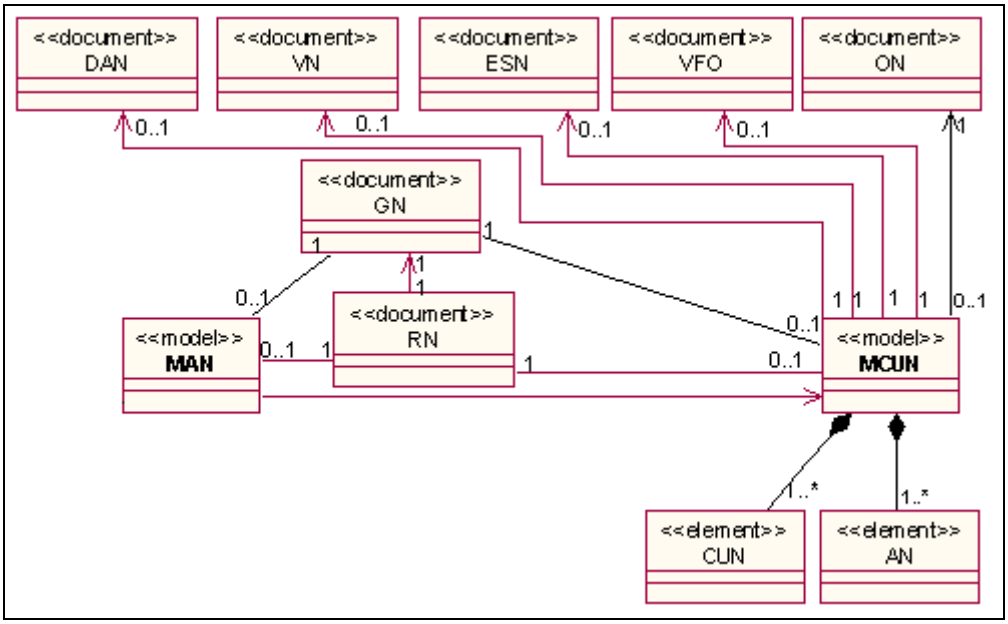


Figura 29: Esquema de Clase de las Reglas de Negocio – RN

Si bien existe un esquema de clase para cada uno de las clases estereotipadas <<document>> del diagrama de clases original (ESN, ON, DAN, VFO, VN), no se representa su traducción a Object-Z en este trabajo, dado que todas las relaciones que tienen con otras clases poseen navegabilidad dirigidas hacia ellas, y por lo tanto sus correspondientes esquemas de clases carecen de atributos específicos, en cuanto a las relaciones con otras clases.

En la figura 30 se muestra la formalización del artefacto MCUN, para lo cual se transcribe la porción del diagrama de clases original, a los fines de visualizar el artefacto MCUN y sus relaciones.

Para el MCUN se define un atributo que indica el estereotipo de la clase model, un atributo para cada una de las clases estereotipadas con <<document>> en el diagrama original, y además dos atributos para representar los elementos del tipo CUN y AN. Estos atributos se definen como conjunto de elementos y contienen el símbolo © para indicar que son partes de una relación de composición hacia el todo, en este caso son las partes de MCUN. Se incluyen restricciones para traducir la multiplicidad de cada una relaciones entre las clases, y además todas las restricciones correspondientes a la navegabilidad de las relaciones de asociación de MCUN a los otros artefactos.



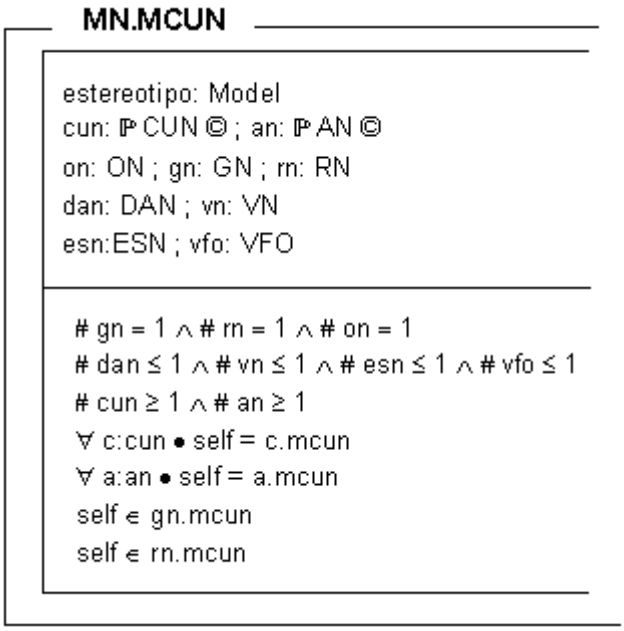


Figura 30: Esquema de Clase del Modelo de Casos de Uso de Negocio - MCUN

CUN contiene un atributo que indica el estereotipo al que corresponde, y el otro atributo surge porque CUN es parte del MCUN. Un predicado formaliza esta relación de composición. Además, CUN es la clase padre de las clases CUNAbstracto y CUNNoAbstracto, por lo tanto en los esquemas de clase que las representan se indica dicha relación de generalización, como se muestra en la figura 31.

La relación a sí misma que se visualiza en el diagrama UML entre elementos de CUN, es formalizada de acuerdo a la descripción dada en el mecanismo de traducción anteriormente definido del tipo self ∈ cun1.cun2. Además, existen dos reglas adicionales, nombradas MCUN.8 y MCUN.9 y definidas en el capítulo 4, que deben ser formalizadas en esta instancia y también aparecen en el esquema de clase de CUN de la figura 31.

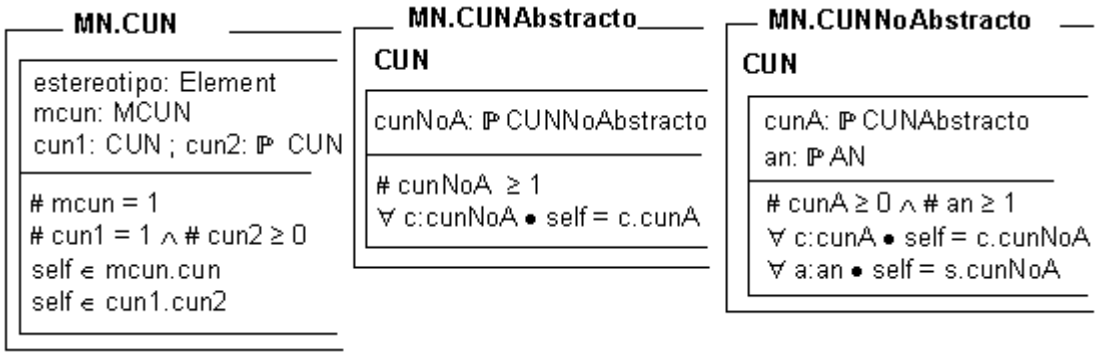


Figura 31: Esquema de Clase de los Casos de Uso de Negocio - CUN

En el caso de la clase AN, al igual que CUN posee los atributos indicando el estereotipo, la relación con MCUN y la relación con la clase CUNNoAbtracto. Además, una formalización importante que aparece en esta clase es la relación a si misma entre actores indicando que puede existir relación entre actores.

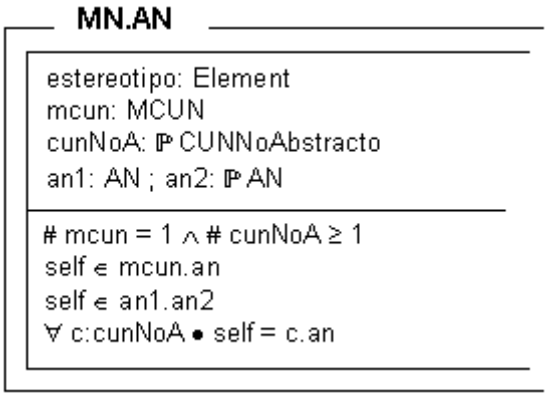


Figura 32: Esquema de Clase de los Actores de Negocio - AN

En la formalización del Modelo de Análisis de Negocio, nombrado MAN, se define un atributo para indicar el tipo de artefacto, un atributo para cada uno de los documentos a los que el MAN tiene visibilidad, y un atributo para cada uno de los artefactos que lo componen, que en el modelo genérico están estereotipados con <<Element>>. Cada atributo que pertenece a un elemento de la composición, es definido como un conjunto de elementos y tienen el símbolo ©, para indicar que son las partes de un todo, rol jugado por MAN en este caso. (Figura 33).

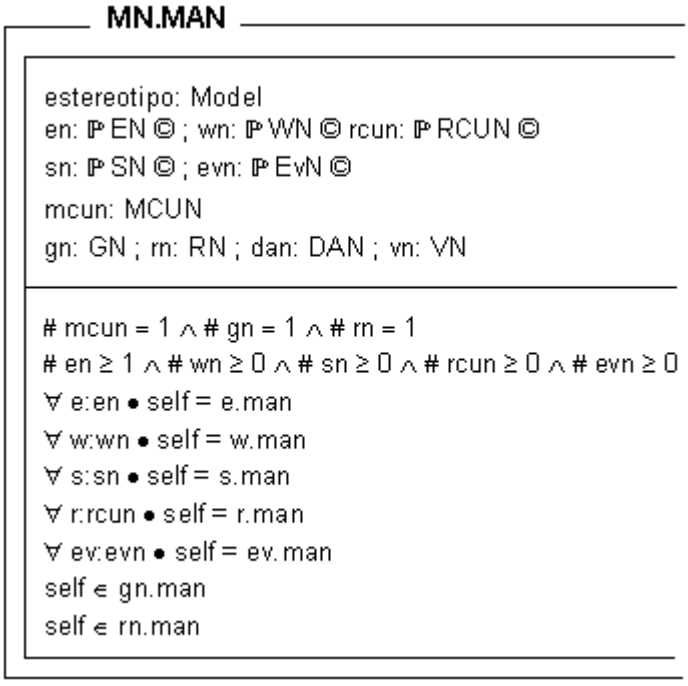


Figura 33: Esquema de Clase del Modelo de Análisis de Negocio - MAN

Además, el MAN depende estructural y semánticamente del MCUN, estructural porque solo puede ser creada una instancia de MAN si por lo menos existe una instancia de MCUN, y semántica porque la creación de cada uno de los elementos del MAN dependen de CUN y AN, que son los elementos del MCUN. En esta formalización se agrega un atributo más en el MAN para indicar la relación con MCUN.

Y al igual que en las formalizaciones ya mostradas, se definen en el MAN las restricciones correspondientes a las asociaciones con otras clases, la multiplicidad y la relación de composición con sus elementos.

Las Entidades del Negocio (EN) son un artefacto muy importante en el modelado del negocio. En el diagrama de clases original de la Figura 17, las EN son una parte del MAN y se modelan con una relación de composición. Es de apreciar que, por la multiplicidad expresada en dicha relación, el MAN posee una o muchas EN asociadas pero no ninguna, y esto se expresa en las restricciones escritas en la formalización de este artefacto. (Figura 34).

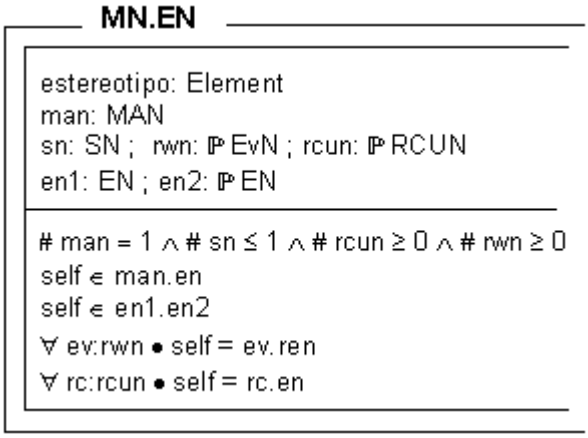


Figura 34: Esquema de Clase del Modelo de Entidades de Negocio - EN

Otra relación importante a visualizar es la que aparece entre las EN y los WN, que modela la existencia de una interacción entre ellos, dada por los eventos del negocio (EvN) que ocurren. Los EvN son parte del MAN, pero además son modelados como una clase asociación entre EN y WN y traducidos a Object-Z (Figura 35), tal como se indicó en el mecanismo de traducción descrito anteriormente. Se escribe la restricción *self ∈ en.wn ∧ self ∈ wn.en para EvN* como para modelar la clase asociación.

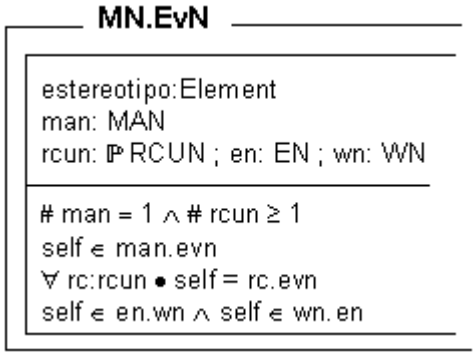


Figura 35: Esquema de Clase de Eventos de Negocio - EN

En la figura 36 se muestra la formalización de la clase WN, donde al igual que para la formalización de las EN (Figura 34), es de notar que se define el atributo *ren:ℙEvN* para la clase asociación EvN y se escribe la restricción $\forall ev{:}ren \bullet self = ev.rwn$. Esto indica que la clase asociación es traducida a una relación ternaria entre EN, WN y EvN, y es tratada como asociaciones comunes entre dichas clases.

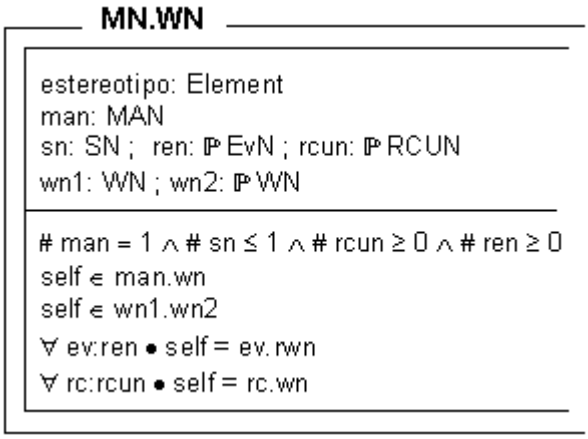


Figura 36: Esquema de Clase de Workers de Negocio - WN

No existe ningún caso particular en la formalización de la clase RCUN más que las consideraciones tenidas en cuenta para los artefactos formalizados anteriormente, igualmente la formalización completa de RCUN es mostrada en la figura 37.

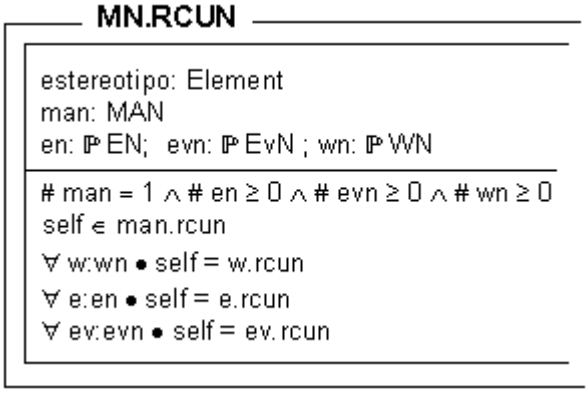


Figura 37: Esquema de Clase de la Realización de Casos de Uso de Negocio - RCUN

SN es uno de los elementos que conforman el MAN. Un Sistema de Negocio encapsula un conjunto de roles y recursos que juntos completan un propósito específico, y define un conjunto de responsabilidades con las cuales el propósito puede conseguirse. El propósito fundamental es reducir la complejidad de las interdependencias e interacciones dentro del negocio. Los diseñadores del negocio usan los Sistemas de Negocio (SN) para formar colecciones de workers (WN) y entidades (EN) relacionados y definir explícitamente dependencias dentro de la organización.

El diagrama de clases de la figura 17 muestra que SN es parte del MAN con una relación de composición, y además posee una relación de agregación simple con EN y WN, por la definición dada. En la figura 38 se muestra la formalización de SN y como puede verse la relación de agregación simple es tratada como a cualquier otra asociación con multiplicidad de 0..1 en SN y 1..n para EN y WN.

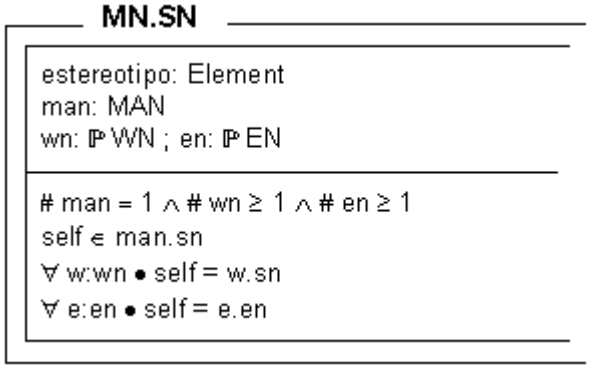


Figura 38: Esquema de Clase de los Sistemas de Negocio - SN

En esta sección se presentó la formalización de cada una de las clases que conforman el conjunto completo de artefactos del modelo de negocio, de acuerdo a las particularidades introducidas por el modelo genérico propuesto (capítulo 4). Falta introducir la formalización de las reglas adicionales propuestas en lenguaje natural, y son presentadas en la siguiente sección.

5.4. Formalización de reglas adicionales

Tal como se introdujo en el capítulo previo, para la definición completa del modelo genérico del modelo de negocio, es necesario un conjunto de reglas adicionales al diagrama de clases de la figura 17. Las reglas fueron descritas en lenguaje natural y en esta sección son formalizadas en Object-Z.

Una vez formalizado el modelo genérico completo, es posible asegurar que la solución propuesta al modelado de negocio representa una propuesta tentadora para los ingenieros de software, que les ayudará tanto a la construcción correcta de un modelo de negocio como a la verificación de un modelo de negocio particular previamente construido.

A continuación, se transcriben las reglas en lenguaje natural y se muestra su traducción a expresiones formales Object-Z. De las reglas definidas (sección 4.3-Capítulo 4), algunas se traducen directamente desde el diagrama de clases y aparecen en los esquemas de clase de la sección previa, mientras que otras reglas corresponden a restricciones adicionales al diagrama y su formalización se expresa en esta sección.

5.4.1. Reglas generales definidas para el Modelo de Negocio (MN)

- (regla MN.1)

MN: {artefacto}. El Modelo de Negocio está compuesto por un conjunto de artefactos.

Se define artefacto como un tipo libre que toma alguno de los valores definidos.

artefacto::= MCUN | MAN | CUN | AN | SN | EN | WN | RCUN | EvN | GN | RN | ON | ESN | DAN | VN | VFO

$\forall$ mn: MN $\bullet$ mn: P artefacto

- (regla MN.6)

Un MN particular tiene a lo sumo una instancia de cada artefacto document.

$\forall$ mn:MN, a: document $\bullet$ a $\in$ mn $\wedge$ (a = ESN $\vee$ a = DAN $\vee$ a = VN $\vee$ a = VFO) $\Rightarrow$ # a $\leq$ 1

- (regla MN.7)

Todo MN particular debe contener un GN, un ON, un RN y al menos un MCUN.

$\forall$ mn:MN $\bullet$ ($\exists$ gn:GN, on:ON, rn:RN, mcun:MCUN $\bullet$ # mn.gn = 1 $\wedge$ # mn.on = 1 $\wedge$ # mn.rn = 1 $\wedge$ # mn.mcun $\leq$ 1)

- (regla MN.8)

Se crea una instancia de MCUN, si ya existe una instancia de GN, RN y ON.

$\forall$ mn:MN, mcun:MCUN, gn:GN, on:ON, rn:RN • (# mn.mcun = 1 • # mn.gn = 1 $\wedge$ # mn.on = 1 $\wedge$ # mn.rn = 1)

- (regla MN.9)

  Se crea una instancia de MAN, si ya existe una instancia de MCUN, GN y RN.

$\forall$ mn: MN, man:MAN, mcun:MCUN, gn:GN, rn:RN • (# mn.man = 1 • # mn.mcun = 1 $\wedge$ # mn.gn = 1 $\wedge$ # mn.rn = 1)

- (regla MN.10)

  Cada regla de negocio definida en RN debe trasladarse a algún elemento de MCUN y/o de MAN.

$\forall$ mn:MN, rn:RN • ($\exists$ e:element | e $\in$ mn.mcun $\vee$ e $\in$ mn.man • mn.rn $\subset$ e)

- (regla MN.11)

  Cada término del GN debería estar incluido en la descripción de por lo menos un CUN.

GN = {(term,descripción)}

CUN = {(especificación,descripción)}

$\forall$ mn:MN, gn:GN, cun:CUN • (t $\in$ dom mn.gn $\Rightarrow$ t $\in$ ran mn.cun)

- (regla MN.12)

  Cada término definido en el GN que representa una entidad (EN), se corresponde con una clase del modelo de dominio.

GN = {(term,descripción)}

md = {EN}

$\forall$ mn:MN, gn:GN, en:EN • (t $\in$ dom gn $\wedge$ t $\in$ mn.en $\Rightarrow$ t $\subset$ md)

5.4.2. Reglas definidas para el Modelo de Casos de Uso de Negocio (MCUN) y sus componentes

- (regla MCUN.4)

  Una instancia de CUN no puede estar relacionada a si misma.

$\forall$ mn :MN, cun1,cun2: CUN • cun1 $\in$ mn $\wedge$ cun2 $\in$ mn $\wedge$ # cun1.cun2 > 0 $\vee$ # cun2.cun1 > 0 $\Rightarrow$ cun1 $\neq$ cun2

- (regla MCUN.7)

  Una instancia de AN no puede estar relacionada a si misma.

  $\forall$ mn :MN, cun1,cun2: CUN • cun1 $\in$ mn $\wedge$ cun2 $\in$ mn $\wedge$ # cun1.cun2 > 0 $\vee$ # cun2.cun1 > 0 $\Rightarrow$ cun1 $\neq$ cun2

5.4.3. Reglas definidas para el Modelo de Análisis de Negocio (MAN) y sus componentes

- (regla MAN.4)

  Cada instancia de WN, EN y EvN deben participar en al menos una instancia de RCUN.

  . $\forall$ mn:MN, wn:WN, en:EN, evn:EvN • wn, en, evn $\in$ mn $\wedge$ ($\exists$ rcun:RCUN • rcun $\in$ mn $\wedge$ # wn.rcun > 0 $\wedge$ # en.rcun > 0 $\wedge$ # evn.rcun > 0)

- (regla MAN.9)

  Cada clase definida como EN debe documentarse en el GN.

  . $\forall$ wn:WN, en:EN, evn:EvN • ($\exists$ rcun:RCUN • # wn.rcun > 0 $\wedge$ # en.rcun > 0 $\wedge$ # evn.rcun > 0)

De esta manera se traducen a Object-Z las reglas que no están implícitas en el diagrama de clases y que son factibles de ser traducidas a un lenguaje formal.

CAPITULO 6: Conclusiones y Trabajos Futuros

Las diferentes etapas que los ingenieros de software construyen para desarrollar un sistema, transitan en torno a las funcionalidades identificadas en la etapa de captura de requerimientos, y refinadas en cada iteración. El problema central radica en determinar de manera correcta las funcionalidades del sistema, necesarias para soportar la estructura y dinámica de la organización donde el sistema funcionará.

El proceso unificado de Rational [RUP] sugiere un método de desarrollo de software que consiste en la creación de diferentes modelos en cada etapa del ciclo de vida de software: modelo de negocio, modelo de casos de uso, modelo de análisis, modelo de diseño, modelo de despliegue, modelo de implementación y modelo de prueba. Cada modelo construido contempla algunos aspectos del sistema y corresponden a diferentes niveles de abstracción. Mientras que en las primeras etapas del ciclo de vida se construyen modelos más abstractos, en etapas posteriores son refinados y sustituidos por modelos cada vez más concretos. Las relaciones entre los modelos pueden verse en una dirección vertical, donde cada modelo está formado por otros y conforman una visión completa del sistema, o en dirección horizontal representando la evolución del modelo, por ejemplo del modelo de negocio al modelo de casos de uso del sistema.

Modelar es una tarea esencial para describir, entender y expresar la esencia de un problema en forma clara y precisa. A menudo sucede que los desarrolladores de software dedican poco tiempo al estudio y análisis del dominio del problema. A medida que los sistemas se tornan más complejos, la captura de los requerimientos del sistema se hace más importante y difícil de definir.

Una razón muy común por la cual algunos desarrolladores no realizan modelos antes de construir, es que existen muchos sistemas que poseen inicialmente una aparente facilidad de solución. Pero en el mundo del software, cuando estos sistemas son implementados e implantados van creciendo en complejidad y los desarrolladores deben adaptarlos en respuesta a nuevos requerimientos. En otros casos, los desarrolladores no eligen modelar simplemente porque no perciben la necesidad de hacerlo, y generalmente cuando se dan cuenta es demasiado tarde.

Aplicar técnicas de modelado requiere de un entrenamiento adicional y del uso de herramientas que repercuten en el tiempo, costo y esfuerzo para el ciclo de vida del desarrollo de un proyecto. Muchos expertos argumentan que la resistencia al

modelado del software es una cuestión meramente cultural. Los desarrolladores tradicionales están particularmente ansiosos en comenzar a codificar inmediatamente cuando se comienza el desarrollo. En la actualidad, esta concepción está cambiando paulatinamente, y ya muchos desarrolladores no conciben la idea del desarrollo de un software sin los modelos abstractos necesarios que lo acompañen durante todo el ciclo de vida.

En el capítulo 3 de este trabajo se expresa claramente la importancia del modelado del negocio como punto de partida para el desarrollo de un sistema. El RUP [RUP] propone un workflow para definir el modelo de negocio (Figura 15) y un conjunto de artefactos (Figura 16) que deben construirse para modelarlo, pero no define una forma sistemática y simplificada que permita relacionar y priorizar los artefactos a construir en casos particulares.

En esta tesis se construyó un modelo genérico del modelo de negocio, sobre la base de la propuesta original del RUP, con el fin de organizar los artefactos a través de relaciones y reglas y sugerir un método que ayude a los desarrolladores de software a identificar y definir los artefactos de un modelo de negocio particular, reduciendo problemas de ambigüedad en la definición y principalmente reduciendo el tiempo insumido por el desarrollador para analizar y priorizar los artefactos a construir para cada caso.

El desarrollador de software puede utilizar el modelo genérico o metamodelo propuesto para instanciarlo con los datos de un negocio particular y obtener su modelo específico. O puede utilizarlo para determinar si un modelo de negocio previamente construido cumple con las relaciones y reglas impuestas por el modelo genérico.

Las actividades realizadas y los resultados obtenidos son descriptos con mayor detalle a continuación:

- **Estudiar el modelo de negocio de RUP**: En primer lugar, se investigaron y encontraron diversos trabajos que utilizan la técnica de modelado de negocio propuesta por RUP [RUP]. No se encontró ninguna propuesta que relacione los artefactos de manera más simplificada, fácil de interpretar y aplicar. En el capítulo 1 (sección 1.3.2) se expuso una breve reseña de los trabajos consultados más importantes.

- **Construir el modelo genérico usando el lenguaje gráfico estándar UML**: Para la construcción del modelo genérico se definieron cada uno de los artefactos como una clase y se identificaron las relaciones entre ellos. Para facilitar la visualización y comprensión del modelo genérico se utilizó el diagrama de clases de UML que permite modelar una vista estática y lógica del problema, y especificar y

documentar modelos estructurales. El modelo genérico o metamodelo propuesto facilita al desarrollador la construcción de un modelo de negocio específico y con las ventajas de contar con una solución prolija y correcta.

- **Definir reglas suplementarias**: el lenguaje de modelado gráfico UML no permite expresar con claridad y sin ambigüedad ciertas condiciones o restricciones que deben imponerse al modelo. Por ello, fue necesario definir reglas adicionales que complementen al modelo genérico con el objeto de construir un modelo de negocio particular de manera más fácil y más correcta.

- **Probar el modelo genérico con un caso de estudio**: se expresó la narrativa para un caso de estudio e instanció el modelo genérico, obteniendo un modelo de negocio particular y además teniendo en cuenta las reglas adicionales. El modelo genérico impone la definición de algunos artefactos en construcción de cualquier modelo de negocio específico.

- **Seleccionar un lenguaje de especificación formal**: para la selección de un lenguaje formal se encontraron y analizaron básicamente varias propuestas relacionadas con el lenguaje formal Object-Z (mencionados en la sección 5.1.) y una propuesta basada en el lenguaje formal RSL [FG03]. A partir de éste análisis se concluyó con Object-Z por varias razones: principalmente porque proporciona un formalismo y una semántica uniforme para la representación de modelos diagramáticos, tales como los diagramas de clase UML, y preserva la mayoría de las características de las estructuras orientadas a objetos de estos modelos semi formales. Además, por la diversidad de trabajos encontrados que relacionan UML con Object-Z permiten suponer que los modelos construidos serán más accesibles y fáciles de interpretar por los ingenieros de Software, que otros lenguajes de especificación formal.

- **Construir un mecanismo de traducción del lenguaje gráfico al lenguaje formal**: Se encontraron numerosos trabajos (ver sección 5.1) que proponen la traducción del lenguaje gráfico a algún lenguaje de especificación formal, y a partir de las ideas de algunos de ellos se construyó, un mecanismo de traducción (ver subsección 5.2.1) que permite traducir el modelo genérico del modelo de negocio representado con UML al lenguaje de especificación formal Object-Z.

- **Formalizar el modelo genérico en el lenguaje de especificación formal Object-Z**: Si bien UML es una poderosa herramienta que permite modelar gráficamente sistemas orientados a objetos, fácil de usar, entender y compartir entre desarrolladores, no posee una semántica precisa y se presentan ambigüedades que originan problemas de interpretaciones erróneas o diferentes. Con las especificaciones formales se logra expresar sistemas con una sintaxis y semántica precisa, complementando las técnicas de especificación informal. Se utilizó el mecanismo de traducción descrito y se tradujo íntegramente el modelo genérico al lenguaje Object-Z, tanto el diagrama de clases como las reglas adicionales definidas. La formalización del modelo genérico del modelo de negocio permite, principalmente, construir herramientas automáticas que faciliten el chequeo de modelos de negocio.

Algunos posibles trabajos futuros se manifiestan brevemente a continuación:

- **Construir un Profile UML del modelo de negocio basado en el modelo genérico propuesto**. La definición de profiles UML constituye el principal mecanismo que este lenguaje ofrece para extender su sintaxis y su semántica y adaptar los modelos UML a un dominio concreto de aplicación. En este sentido, el modelo genérico del modelo de negocio cumple con este propósito. De acuerdo a la especificación de la infraestructura de UML 2.0 [UML2] las diferentes razones por las que un desarrollador querría adaptar un metamodelo se pueden resumir en: disponer de una terminología propia para un dominio particular, definir una sintaxis para construcciones que no cuentan con una notación propia, definir una notación diferente para símbolos ya existentes, agregar semántica, añadir restricciones que limiten la forma de usar el metamodelo y sus constructores o añadir información que puede ser útil cuando se transforma un modelo en otros modelos, o a código.

- **Desarrollar una herramienta que automatice el chequeo de relaciones entre artefactos y las reglas formalizadas con Object-Z.** Una de principales razones por las que se realizó la formalización del modelo genérico fue contar con una definición precisa del modelo y permitir la construcción de una herramienta que tome un modelo de negocio particular como entrada y retorne un reporte con las características del mismo, teniendo en cuenta las reglas establecidas para la construcción de un modelo de negocio correcto según el modelo genérico propuesto en este trabajo.

Además, también se prevé proponer una extensión de una herramienta open source utilizada para graficar diagramas de UML e incorporarle las restricciones del modelo de negocio según se definió en este trabajo.

- **Evolucionar con una propuesta genérica a los siguientes modelos del desarrollo.** De la misma manera que se consiguió definir el modelo genérico del modelo de negocio, se podrían construir modelos genéricos para cada uno de los modelos sugeridos para las siguientes etapas, considerando en cada caso que un modelo resulta de la evolución horizontal del modelo o los modelos construidos en la etapa anterior.

Referencias Bibliográficas

[ABL96] J-R Abrial, E. Börger, H. Langmaack. Formal Methods for Industrial Appications, Specifying and Programming the Steam Boiler Control. ISBN: 3-540-61929-1 Springer-Verlag, 1996.

[AP03] N. Amálio, F. Polack. Comparison of Formalisation Approaches of UML Class Constructs in Z and Object-Z. In International Conference of Z and B Users (ZB 2003), volume 2561 of Lecture Notes in Computer Science. Springer-Verlag, 2003.

[ASP04] N. Amálio, Susan Stepney, F. Polack. Fromal Proof from UML Models, *ICFEM'04, Seattle, USA, 2004*, pp 418-433. LNCS 3308. Springer, 2004.

[BDMN04a] G. Baum, M. Daniele, P. Martellotto, M. Novaira. Propuesta de un Modelo Genérico para el Modelo de Negocio. VI Workshop de Investigadores en Ciencias de la Computación (WICC 2004), Mayo 2004, Universidad Nacional de Comahue, Argentina

[BDMN04b] G. Baum, M. Daniele, P. Martellotto, M. Novaira. Un Modelo Genérico para el Modelo de Negocio. X Congreso Argentino de Ciencias de la Computación, CACIC'2004: Universidad Nacional de La Matanza, Argentina, Octubre 2004.

[BP03] Becker, Pons C. Definición Formal de la Semántica de UML-OCL a través de su traducción a Object-Z. CACIC 2003.

[BRJ99] Grady Booch, James Rumbaugh, Ivar Jacobson. The Unified Modeling Language. Addison Wesley, 1999.

[DMB05a] Daniele M., Martellotto P., Baum G. Traducción del Modelo Genérico del Modelo de Negocio a Objetc-Z. Publicado en los anales del VII Workshop de Investigadores en Ciencias de la Computación (WICC 2005), Universidad Nacional de Río Cuarto, Argentina, Mayo 2005.

[DMB05b] Daniele M., Martellotto P., Baum G. Formalización del Modelo Genérico del Modelo de Negocio, publicado en los anales de la 34 JAIIO (Jornadas Argentinas de Informática e Investigación Operativa), en el marco del Simposio ASIS (Simposio Argentino en Sistemas de Información), Septiembre 2005, Rosario, Argentina.

[Dapena02] Delgado Dapena, Martha D. Definición del modelo del negocio y del dominio utilizando Razonamiento Basado en Casos, La Revista Electrónica del DIICC, ISSN : 0717 – 4195, Edición 8, 2002.

[DeMarco79] Tom DeMarco. Libro: Structured Analysis and System

Specification, 1979.

[DMN03] Daniele,M, Martellotto, P., Novaira, M. Informe del Modelo de Negocio. Artefactos, 2003. Reporte técnico. http://dc.exa.unrc.edu.ar/investigacion/index.html

[DMN04] Daniele,M., Martellotto, P., Novaira, M. Definición de Reglas para los Artefactos del modelo de negocio, 2004. Reporte técnico. http://dc.exa.unrc.edu.ar/investigacion/index.html

[EP00] Eriksson, H.E., Penker, M. Business Modeling with UML. Business Patterns at Work. John Wiley & Sons, Inc. 2000. ISBN:0-471-29551-5.

[FV04] Fuentes L., Vallecito A. Una Introducción a los Perfiles UML, 2004.http://www.lcc.uma.es/~av/Publicaciones/04/UMLProfiles-Novatica04.pdf

[FG03] Funes, Ana M., George Chris. Chapter VIII: Formalizing UML Class Diagram. Del libro UML and Unified Process de Liliana Favre. ISBN: 1-931777-44-6. eISBN: 1-931777-60-8.

[GB04] Geoffrey Bessin, Market Manager, Software Quality Products, IBM Rational. The business value of software quality.15/Jun/2004. http://www-128.ibm.com/developerworks/rational/library/4995.html

[GJM91] Ghezzi Carlo, Jazayeri Mehdi, Mandrioli Dino, Fundamentals of Software Engeneering, Prentice-Hall Inc., New Jersey, 1991

[Heumann03] Jim Heumann. Introduction to business modeling using the Unified Modeling Language (UML). IBM. 16 June 2003. http://www-106.ibm.com/developerworks/rational/library/360.html

[Jacobson92] Ivar Jacobson y otros. Object Oriented Software Engineering. A Use Case Driven Approach. Addison Wesley, 1992.

[JBR99] Ivar Jacobson, Grady Booch, James Rumbaugh. “The Unified Software Development Process”. Addison Wesley, 1999.

[KC99] S.-K. Kim amd D. Carrington. Formalising the UML class diagram using Object-Z. In *2nd International Conference on Unified Modelling Language (UML'99)*, volume 1732 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.

[KC00] S-K. Kim and D. Carrington. A Formal Mapping between UML Models and Object-Z Specifications. In ZB2000: International Conference of B and Z Users, volume 1878 of Lecture Notes in Computer Science, 2000.

[KC01] S-K. Kim, D. Carrington and Roger Duke. A Metamodel-based transformation between UML and Object-Z. 0-7995-0474-4/01. IEEE, 2001.

[KC04] Kim and Carrington. A Formal Object-Oriented Approach to defining Consistency Constraints for UML Models, in Proc. of

Australian Software Engineering Conference (*ASWEC'2004)*, Melbourne, Australia, Apr. 2004, IEEE Computing Society.

[KM97] C. Klauck and H.-J. Mueller. Formal business process engineering based on graph grammars. International Journal on Production Economics, 50:129–140, Special Issue on Business Process Reengineering, 1997

[KP00] M. Koubarakis and D. Plexousakis. A formal model for business process modeling and design. In Conference on Advanced Information Systems Engineering, pages 142–156, 2000.

[MBM03] P. Moura, R. Borges, and A. Mota. Experimenting Formal Methods through UML. Submitted to WMF'2003.

[Meyer97] Meyer, Bertrand. Object Oriented Software Construction. Second Edition. Published by Prentice Hall PTR, 1997.

[NAUR69] Naur, P. y B. Randall (eds.) Software Engineering: A Report on a Conference Sponsored by the NATO Science Committee, NATO, 1969.

[OCL01] OMG Unified Modeling Language Specification, Version 1.4, September 2001. http://www.omg.org/library/issuerpt.htm. Capítulo 6: Object Constraint Language Specification

[OMG] OMG Object Management Group. http://www.omg.com/.

[OMMN01] M. Ortín, J. García Molina, B. Moros, J. Nicolás, "El Modelo del Negocio como base del Modelo de Requisitos". 2001. http://www.lsi.us.es/~amador/JIRA/Ponencias/JIRA_Ortin.pdf.

[Pressman01] Pressman, Roger, "Software Engineering: A Practioner´s Approach". Fifth Edition. McGraw-Hill, Inc., 2001.

[RBR03] D. Roe, K. Broda, A. Russo. Mapping UML Models incorporating OCL Constraints into Object-Z. Technical Report 2003/9, Imperial college London.

[RUP] Rational Unified Process. http://www.rational.com/rup/

[Salm03] José Francisco Salm Junior : "*Extensões Da Uml Para Descrever Processos De Negócio*". Florianópolis, Janeiro De 2003.

[SVC01] Pedro Sinogas, André Vasconcelos, Artur Caetano, João Neves, Ricardo Mendes, José Tribolet. Business Processes Extensions to UML Profile for Business Modeling. 2001. Http://www.inesc-id.pt/pt/indicadores/Ficheiros/894.pdf

[Smith95] Graeme Smith. Extending W for Object-Z. ZUM'95, Springer, 1995.

[Smith00] Graeme Smith. The Object-Z Specification Language. Advances in Formal Methods. Kluwer Academic Publishers, 2000.

[Sparks] Geoffrey Sparks, Sparx Systems, Australia. Una Introducción al UML. El Modelo de Proceso de Negocio. Traducción: F. Pinciroli (Solus S.A., Argentina) y A. Orlic (Craftware, Chile).

[Spivey92] Spivey J.M (1989 & 1992) The Z Notation: A Reference Manual, Prentice Hall. http:/Spivey.oriel.ox.ac.uk/~mike/zrm/.

[Suppe77] Frederick Suppe. The Structure of Scientific Theories. Illini Books edition, 1977.

[SVC01] Pedro Sinogas, André Vasconcelos, Artur Caetano, João Neves, Ricardo Mendes, José Tribolet. Business Processes Extensions to UML Profile for Business Modeling. 2001. http://www.inesc-id.pt/pt/indicadores/Ficheiros/894.pdf

[UML1] Unified Modeling Language Specification. http://www.omg.org/technology/documents/vault.htm#modeling

[UML2] UML 2.0.Especificación adoptada por OMG Object Management Group.http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML

Anexo I: El Proceso Unificado

El Proceso Unificado [JBR99] [RUP] es una metodología de desarrollo de software que define *quién* está haciendo *qué*, *cuándo*, y *cómo* para construir o mejorar un producto de software. Es una guía para todos los participantes del proyecto: clientes, usuarios, desarrolladores, directivos. Ordena las actividades del equipo. Dirige las tareas de cada desarrollador y del equipo como un todo, y especifica los artefactos que deben desarrollarse. Además, ofrece criterios para el control y la medición, reduce riesgos y hace el proyecto más predecible.

El Proceso Unificado utiliza UML [BRJ99], como medio de expresión de los diferentes modelos que se crean durante las etapas del desarrollo. UML es un lenguaje estándar de modelado que permite visualizar, especificar, construir y documentar los artefactos de un producto de software.

El Proceso Unificado está dirigido por casos de uso

Para construir un sistema es primordial conocer lo que sus futuros usuarios necesitan y desean. El término usuario representa alguien o algo (como otro sistema fuera del sistema en consideración) que interactúa con el sistema en desarrollo. Una interacción de este tipo es un caso de uso. Un caso de uso es un fragmento de funcionalidad del sistema que proporciona al usuario un resultado importante. Los casos de uso representan los requisitos funcionales. Todos los casos de uso juntos constituyen el modelo de casos de uso, el cual describe la funcionalidad total del sistema. Basándose en el modelo de casos de uso, los desarrolladores crean una serie de modelos de diseño e implementación que llevan a cabo los casos de uso.

El Proceso Unificado está centrado en la arquitectura

La arquitectura de un sistema software se describe mediante diferentes vistas del sistema en construcción. Se ve influida por muchos factores, como la plataforma en la que tiene que funcionar el software (arquitectura hardware, sistema operativo, sistema de gestión de base de datos, protocolos para comunicaciones en red…), consideraciones de implantación, requisitos no funcionales (rendimiento, fiabilidad). La arquitectura es una vista del diseño completo con las características más importantes resaltadas, dejando los detalles de lado.

Debe haber interacción entre los casos de uso y la arquitectura. Los casos de uso deben encajar en la arquitectura cuando se llevan a cabo, y la arquitectura debe permitir el desarrollo de todos los casos de uso requeridos, ahora y en el futuro.

Los arquitectos modelan el sistema para darle una forma, y para encontrarla, deben trabajar sobre la comprensión general de las funciones clave, es decir, sobre los casos de uso claves del sistema.

El Proceso Unificado es iterativo e incremental

El desarrollo de un producto software supone un gran esfuerzo que puede durar entre varios meses hasta posiblemente un año o más. Es práctico dividir el trabajo en partes más pequeñas o miniproyectos. Cada miniproyecto es una iteración que resulta en un incremento. Las iteraciones hacen referencia a pasos en el flujo de trabajo, y los incrementos, al crecimiento del producto. Para una efectividad máxima, las iteraciones deben estar controladas; esto es, deben seleccionarse y ejecutarse de una forma planificada.

Los desarrolladores basan la selección de lo que se implementará en una iteración en dos factores. En primer lugar, la iteración trata un grupo de casos de uso que juntos amplían la utilidad del producto desarrollado hasta ahora. En segundo lugar, la iteración trata los riesgos más importantes. Las iteraciones sucesivas se construyen sobre los artefactos de desarrollo tal como quedaron al final de la última iteración. Al ser miniproyectos, comienzan con los casos de uso y continúan a través del trabajo de desarrollo siguiente -análisis, diseño, implementación y prueba-, que termina convirtiendo en código ejecutable los casos de uso que se desarrollaban en la iteración.

En cada iteración, los desarrolladores identifican y especifican los casos de uso relevantes, crean un diseño utilizando la arquitectura seleccionada como guía, implementan el diseño mediante componentes, y verifican que los componentes satisfacen los casos de uso. Si una iteración cumple con su objetivo, el desarrollo continúa con la siguiente iteración. Caso contrario, deben revisar sus decisiones previas y probar con un nuevo enfoque.

Para alcanzar el mayor grado de economía en el desarrollo, un equipo de proyecto intentará seleccionar sólo las iteraciones requeridas para lograr el objetivo del proyecto. Intentará secuenciar las iteraciones en un orden lógico. Un proyecto con éxito se ejecutará de una forma directa, sólo con pequeñas desviaciones del curso que los desarrolladores planificaron inicialmente. Por supuesto, en la medida en que se

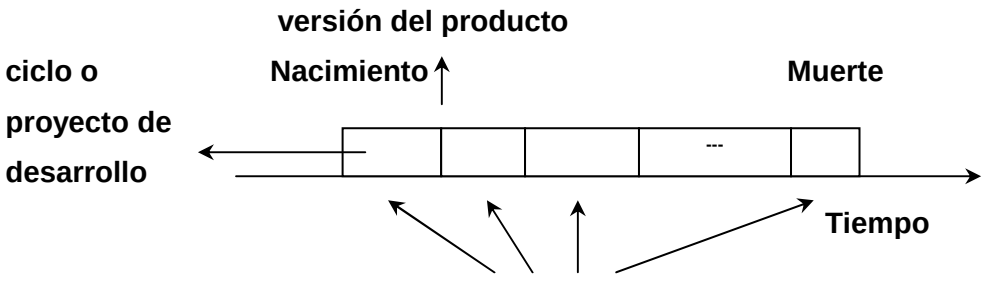
añadan iteraciones o se altere el orden de las mismas por problemas inesperados, el proceso de desarrollo consumirá más esfuerzo y tiempo.

Son muchos los beneficios de un proceso iterativo controlado:

- La iteración controlada reduce el costo del riesgo a los costes de un solo incremento. Si los desarrolladores tienen que repetir la iteración, la organización sólo pierde el esfuerzo mal empleado de la iteración, no el valor del producto entero.
- La iteración controlada reduce el riesgo de no sacar al mercado el producto en el calendario previsto. Mediante la identificación de riesgos en fases tempranas del desarrollo, el tiempo que se gasta en resolverlos se emplea al principio de la planificación. En el método “tradicional”, en el cual los problemas complicados se revelan por primera vez en la prueba del sistema, el tiempo necesario para resolverlos normalmente es mayor que el tiempo que queda en la planificación, y casi siempre obliga a retrasar la entrega.
- La iteración controlada acelera el ritmo del esfuerzo de desarrollo en su totalidad, debido a que los desarrolladores trabajan de manera más eficiente para obtener resultados claros a corto plazo, en lugar de tener un calendario largo, que se prolonga eternamente.
- La iteración controlada reconoce que las necesidades del usuario y sus correspondientes requisitos no pueden definirse completamente al principio. Típicamente, se refinan en iteraciones sucesivas. Esta forma de operar hace más fácil la adaptación a los requisitos cambiantes.
- La arquitectura proporciona la estructura para guiar las iteraciones, mientras que los casos de uso definen los objetivos y dirigen el trabajo de cada iteración.

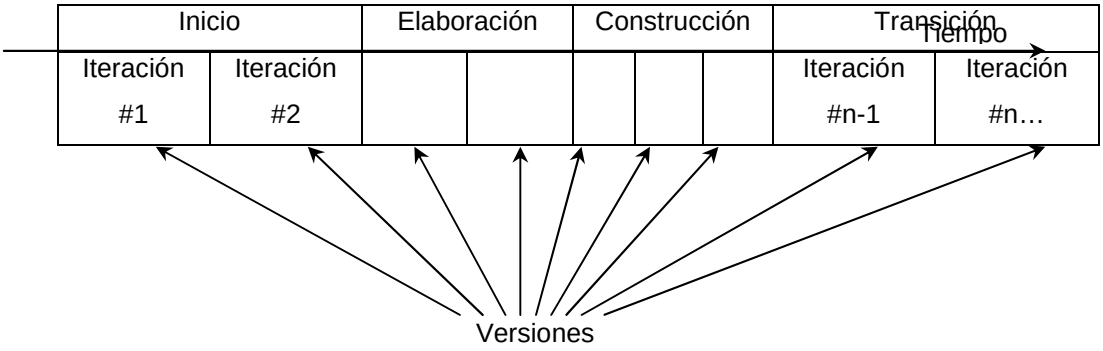
La vida del Proceso Unificado

El Proceso Unificado se repite a lo largo de una serie de ciclos que constituyen la vida de un sistema:



Los ciclos concluyen con una versión

Cada ciclo concluye con una versión del producto para los clientes y consta de cuatro fases: inicio, elaboración, construcción y transición. Cada fase se subdivide a su vez en iteraciones:



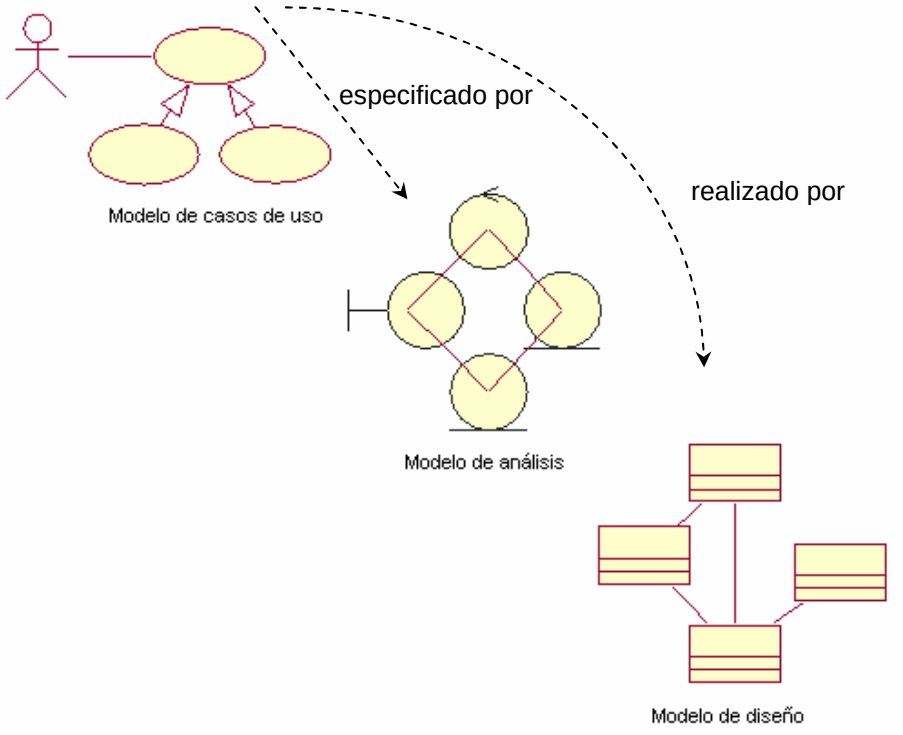
El producto

Cada ciclo produce una nueva versión del sistema, y cada versión es un producto preparado para su entrega. Consta de un cuerpo de código fuente incluido en componentes que puede compilarse y ejecutarse, además de manuales y otros productos asociados.

El producto terminado incluye los requisitos, casos de uso, especificaciones no funcionales y casos de prueba. Aunque los componentes ejecutables sean los artefactos más importantes desde la perspectiva del usuario, no son suficientes por sí solos. Esto se debe a que el entorno cambia. Se mejoran los sistemas operativos, los sistemas de base de datos y las máquinas que los soportan. A medida que el objetivo del sistema se comprende mejor, los propios requisitos pueden cambiar. Para llevar a cabo el siguiente ciclo de manera eficiente, los desarrolladores necesitan todas las representaciones del producto software:

- Un modelo del dominio o modelo del negocio que describa el contexto del negocio en el que se halla el sistema.
- Un modelo de casos de uso, con todos los casos de uso y su relación con los usuarios.
- Un modelo de análisis, con dos propósitos: refinar los casos de uso con más detalle y establecer la asignación inicial de funcionalidad del sistema a un conjunto de objetos que proporcionan el comportamiento.
- Un modelo de diseño que define: (a) la estructura estática del sistema en la forma de subsistemas, clases e interfaces y (b) los casos de uso reflejados como colaboraciones entre los subsistemas, clases e interfaces.

Todos estos modelos están relacionados y representan al sistema como un todo. Los elementos de un modelo poseen dependencias de traza hacia atrás y hacia delante, mediante enlaces hacia otros modelos. Por ejemplo, podemos hacer el seguimiento de un caso de uso (en el modelo de casos de uso) hasta una realización de caso de uso (en el modelo de diseño). La trazabilidad facilita la comprensión y el cambio.



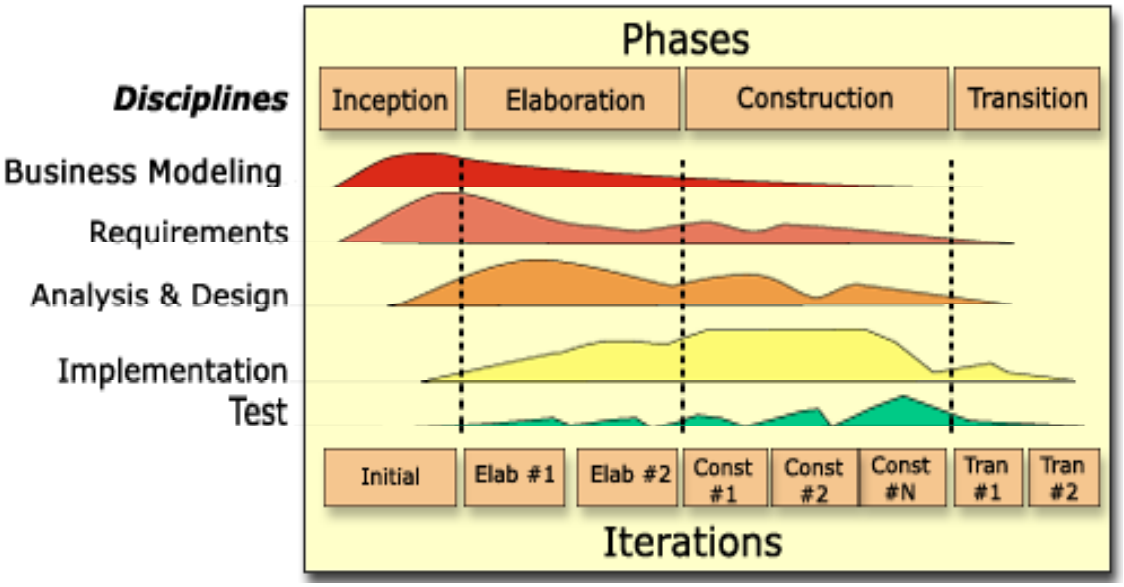
Fases dentro de un ciclo

Cada ciclo se desarrolla a lo largo del tiempo. Este tiempo, a su vez, se divide en cuatro fases y a través de una secuencia de modelos, los desarrolladores visualizan lo que está sucediendo en cada una de ellas. Dentro de cada fase, pueden descomponer adicionalmente el trabajo -en iteraciones con sus incrementos resultantes. Cada fase termina con un hito. Cada hito se determina por la disponibilidad de un conjunto de artefactos.

Los hitos tienen muchos objetivos. El más crítico es que los directores deben tomar ciertas decisiones cruciales antes de que el trabajo pueda continuar con la siguiente fase. Los hitos también permiten, controlar el progreso del trabajo según pasa por esos cuatro puntos clave. Al final, se obtiene un conjunto de datos a partir del seguimiento del tiempo y esfuerzo consumido en cada fase. Estos datos son útiles en

la estimación del tiempo y los recursos humanos para otros proyectos, en la asignación de los recursos durante el tiempo que dura el proyecto, y en el control del progreso contrastado con las planificaciones.

La siguiente figura muestra en la columna izquierda los flujos de trabajo -requisitos, análisis, diseño, implementación y prueba. Las curvas son una aproximación de hasta dónde se llevan a cabo los flujos de trabajo en cada fase. Una iteración típica pasa por los cinco flujos de trabajo.



Durante la fase de inicio, se desarrolla una descripción del producto final a partir de una buena idea y se presenta el análisis de negocio para el producto. Esencialmente, esta fase responde a las siguientes preguntas:

- ¿Cuáles son las principales funciones del sistema para sus usuarios más importantes?
- ¿Cómo podría ser la arquitectura del sistema?
- ¿Cuál es el plan de proyecto y cuánto costará desarrollar el producto?

En esta fase, se identifican y priorizan los riesgos más importantes, se planifica en detalle la fase de elaboración, y se estima el proyecto de manera aproximada.

Durante la fase de elaboración, se especifican en detalle la mayoría de los casos de uso del producto y se diseña la arquitectura del sistema. La relación entre esta última y el propio sistema es primordial. Una manera simple de expresarlo es decir que la arquitectura es análoga al esqueleto cubierto por la piel pero con muy poco músculo (el software) entre los huesos y la piel -sólo lo necesario para permitir que el esqueleto haga movimientos básicos. El sistema es el cuerpo entero con esqueleto, piel, y músculos.

Por tanto, la arquitectura se expresa en forma de vistas de todos los modelos del sistema, que juntos representan al sistema entero. Esto implica que hay vistas arquitectónicas del modelo de casos de uso, del modelo de análisis, y del modelo de diseño.

Al final de la fase de elaboración, el director de proyecto puede planificar las actividades y estimar los recursos necesarios para lkevar a cabo el proyecto.

Durante la fase de construcción, se crea el producto -se añaden los músculos (software terminado) al esqueleto (la arquitectura). La descripción evoluciona hasta convertirse en un producto preparado para ser entregado a la comunidad de usuarios. El grueso de los recursos requeridos se emplea durante esta fase del desarrollo. Al final de esta fase, el producto contiene todos los casos de uso que la dirección y el cliente han acordado para el desarrollo de esta versión. Sin embargo, puede que no esté completamente libre de defectos. Muchos de estos defectos se descubrirán y solucionarán durante la fase de transición.

La fase de transición cubre el período durante el cual el producto se convierte en versión beta. En la versión beta un número reducido de usuarios con experiencia prueba el producto e informa de defectos y deficiencias. Los desarrolladores corrigen los problemas e incorporan algunas de las mejoras sugeridas en una versión general dirigida a la totalidad de la comunidad de usuarios. La fase de transición conlleva actividades como la formación del cliente, el proporcionar una línea de ayuda y asistencia, y la corrección de los defectos que se encuentren tras la entrega. El equipo de mantenimiento suele dividir esos defectos en dos categorías: los que tienen suficiente impacto en la operación para justificar una versión incrementada y los que pueden corregirse en la siguiente versión normal.

Flujo de Trabajo: Captura de requisitos: de la visión a los requisitos

La captura de requisitos es el proceso de averiguar, normalmente en circunstancias difíciles, los requisitos que el sistema a construir debe satisfacer.

El propósito fundamental del flujo de trabajo de los requisitos es guiar el desarrollo hacia el sistema correcto. Esto se consigue mediante una descripción de los requisitos del sistema (es decir, las condiciones o capacidades que el sistema debe cumplir) suficientemente buena como para que pueda llegarse a un acuerdo entre el cliente (incluyendo a los usuarios) y los desarrolladores sobre qué debe y qué no debe hacer el sistema.

Los resultados del flujo de trabajo de los requisitos también ayudan al director del proyecto a planificar las iteraciones y las versiones del cliente.

Para comenzar el desarrollo de un sistema se debe tener en cuenta:

- Enumerar los requisitos candidatos: Son ideas que se les ocurren a los usuarios, clientes, analistas y desarrolladores, que se van a incluir en una lista de requisitos candidatos que se podrían implementar en una versión futura del sistema.
- Comprender el contexto del sistema: Para capturar los requisitos correctos y construir el sistema correcto los desarrolladores requieren un firme conocimiento del contexto en el que se emplaza el sistema. Para lograr este conocimiento, se realiza un modelo del negocio, cuyo objetivo es describir los procesos del negocio con el fin de comprenderlos. A medida que se modela el negocio se comprende el contexto del sistema software. Este modelo establece las competencias requeridas en cada proceso: sus trabajadores, sus responsabilidades, y las operaciones que llevan a cabo. Por supuesto, este conocimiento es decisivo en la identificación de los casos de uso. El modelo del dominio es una parte del modelo de negocio, y describe los conceptos importantes del contexto como objetos del dominio, y los enlaza unos con otros. Además, los objetos del dominio ayudarán a identificar clases a medida que se analiza y diseña el sistema.
- Captura de requisitos funcionales: Los casos de uso especifican funcionalidades que el sistema proporcionará a los usuarios.
- Captura de requisitos no funcionales: Estos requisitos especifican propiedades del sistema, como restricciones del entorno o de la implementación, rendimiento, dependencias de la plataforma, facilidad de mantenimiento, extensibilidad y fiabilidad.

El papel de los requisitos en el ciclo de vida del software

Durante las distintas fases del desarrollo del software, los requisitos más importantes se capturan en la fase de inicio y los restantes se capturan en la fase de elaboración, muy pocos en la fase de construcción. En la fase de transición sólo se cambian los requisitos que así lo requieran.

¿Qué es un modelo del dominio?

Un modelo del dominio captura los tipos más importantes de objetos en el contexto del sistema. Los objetos del dominio representan las cosas que existen o los eventos que suceden en el entorno en el que trabaja el sistema.

Muchos de los objetos del dominio o clases pueden obtenerse mediante una especificación de requisitos. Las clases del dominio aparecen en tres formas típicas:

- Objetos del negocio que representan cosas que se manipulan en el negocio, en nuestro caso por ejemplo tenemos historia clínica, historia clínica resumida, pedido de derivación, pedido de internación, etc.
- Objetos del mundo real y conceptos en los que el sistema debe hacer un seguimiento.
- Sucesos que ocurrirán o han ocurrido como la llegada de un paciente.

El modelo del dominio se describe mediante diagramas de clases de UML. Las clases que no se incluyan en el modelo del dominio se incluirán en un glosario de términos.

¿Qué es un modelo del negocio?

El modelado del negocio es una técnica para comprender los procesos de negocio de la organización. Se lo especifica con el modelo de caso de uso del negocio, que esta compuesto por casos de uso del negocio y actores del negocio que se corresponden con los procesos del negocio y clientes respectivamente. La realización de un caso de uso del negocio puede mostrarse en diagramas de interacción y diagramas de actividad.

Captura de requisitos como casos de uso

El esfuerzo principal en esta fase es desarrollar un modelo de casos de uso del sistema. Los casos de uso proporcionan un medio intuitivo y sistemático para capturar los requisitos funcionales con un énfasis especial en el valor añadido para cada usuario individual o para cada sistema externo.

Artefactos

Artefacto: Modelo de Casos de Uso

El modelo de casos de uso sirve como acuerdo entre clientes y desarrolladores, y proporciona la entrada fundamental para el análisis, el diseño y las pruebas.

Un modelo de casos de uso es un modelo del sistema que contiene actores, casos de uso y relaciones entre ellos. Los elementos de este modelo son obtenidos a partir del modelo de negocio.

Artefacto: Actor

En el modelo de casos de uso los actores representan a los usuarios, en nuestro modelo el actor es el médico cardiólogo. Por tanto, los actores representan terceros fuera del sistema que colaboran con éste. Los casos de uso proporcionan valor al actor.

Un actor juega un papel por cada caso de uso con el que colabora. Cada vez que un usuario interactúa con el sistema, la instancia correspondiente del actor está desarrollando ese papel. Una instancia de un actor es por tanto un usuario que interactúa con el sistema.

Artefacto: Caso de Uso

Cada forma en que los actores usan el sistema se representa con un caso de uso. Los casos de uso son “fragmentos” de funcionalidad que el sistema ofrece para aportar un resultado de valor para sus actores. De manera más precisa, un caso de uso especifica una secuencia de acciones que el sistema puede llevar a cabo interactuando con sus actores, incluyendo alternativas dentro de la secuencia.

Una instancia de caso de uso es la realización (o ejecución) de un caso de uso. Otra forma de decirlo es que una instancia de un caso de uso es lo que el sistema lleva a cabo cuando “obedece a un caso de uso”. Cuando se lleva a cabo una instancia de un caso de uso, ésta interactúa con instancias de actores, y ejecuta una secuencia de acciones según se especifica en el caso de uso. Ésta secuencia pueder ser especificada con un diagrama de estado o un diagrama de actividad.

Flujo de sucesos

El flujo de sucesos para cada caso de uso puede plasmarse como una descripción textual de la secuencia de acciones del caso de uso. Por tanto, el flujo de sucesos especifica lo que el sistema hace cuando se lleva a cabo un caso de uso especifico. El flujo de suceso también especifica cómo interactúa el sistema con los actores cuando se lleva a cabo el caso de uso.

Artefacto: Descripción de la Arquitectura (vista del modelo de casos de uso)

La descripción de la arquitectura contiene una vista de la arquitectura del modelo de casos de uso, que representa los casos de uso significativos para la arquitectura.

La vista de la arquitectura del modelo de casos de uso debería incluir los casos de uso que describan alguna funcionalidad importante y critica, o que impliquen algún requisito importante que deba desarrollarse pronto dentro del ciclo de vida del software.

Artefacto: Glosario

Podemos utilizar un glosario para definir términos comunes importantes que usaremos al describir el sistema. Un glosario es muy útil para alcanzar un consenso entre los desarrolladores relativo a la definición de los diversos conceptos y nociones,

y para reducir en general el riesgo de confusiones. El glosario se comienza en el modelo de negocio y se refina en esta etapa.

Trabajadores

Un trabajador es un puesto al cual se puede asignar una persona real, se puede decir que un trabajador representa una abstracción de un ser humano con ciertas capacidades que se requieren en un caso de uso del negocio.

Trabajador: Analista de Sistemas

El analista de sistemas es el responsable de los requisitos del modelo de casos de uso, los funcionales y no funcionales. También es responsable de delimitar al sistema encontrando los actores y casos de uso asegurándose que el modelo de casos de uso es completo y consistente.

Trabajador: Especificador de Casos de Uso

Es el responsable de la descripción detallada de un caso de uso, y de esta forma asiste la tarea del analista.

Trabajador: Diseñador de Interfaz de Usuario

La tarea de este trabajador es desarrollar prototipos de interfaz de usuario, para algunos casos de uso, habitualmente un prototipo para cada actor.

Trabajador: Arquitecto

Es el responsable de describir la vista de la arquitectura del modelo de casos de uso.

Flujo de trabajo

Las actividades del flujo de trabajo son realizadas por los trabajadores nombrados arriba y estas consisten en la creación y modificación de artefactos. Describimos los flujos de trabajo como una secuencia de actividades que están ordenadas, así que una actividad produce una salida que da entrada a la siguiente actividad.

Actividad: Encontrar Actores y Casos de Uso

Se identifican los actores y casos de uso para:

- Delimitar el sistema de su entorno.
- Especificar quien y qué (actores) interactuarán con el sistema, y qué funcionalidad (casos de uso) se espera del sistema.

La identificación de actores y casos de uso es la actividad más decisiva para obtener adecuadamente los requisitos, y es responsabilidad del analista de sistemas. Para realizar esta actividad, se hace un análisis del modelo del negocio

Encontrar los actores: se definen los actores del sistema a partir de los actores de negocio. Encontrar un nombre relevante para el actor es importante para comunicar la semántica deseada. La descripción breve del actor debe esbozar sus necesidades y responsabilidades.

Encontrar los casos de uso: Como para encontrar casos de uso se parte del modelo del negocio, se propone un caso de uso por cada rol del trabajador que participa en la realización de casos de uso del negocio y que utilizará información del sistema. El actor necesitará normalmente casos de uso para soportar su trabajo de creación, cambio, rastreo, eliminación o estudio de los objetos del negocio. Se elige un nombre para cada caso de uso de forma que haga pensar en la secuencia de acciones concreta que añade valor a un actor.

Describir brevemente cada caso de uso: Cuando se describe un caso de uso se puede comenzar haciéndolo brevemente o solo escribiendo su nombre, pero luego se debe describir en forma textual paso a paso lo que el caso de uso necesita hacer cuando interactúa con sus actores.

El objetivo principal de detallar cada caso de uso es describir su flujo de sucesos en detalle, incluyendo cómo comienza, termina e interactúa con los actores.

La descripción de un caso de uso incluye lo siguiente:

- Definición de un estado inicial o precondición.
- Cómo y cuándo comienza un caso de uso (la primera acción a ejecutar).
- El orden en que se ejecutan el flujo principal de acciones, se definirá cómo una secuencia numerada de pasos.
- Cómo y cuándo termina un caso de uso.
- Definición de los posibles estados finales o poscondiciones.
- La descripción de posibles caminos alternativos al flujo principal de acciones.
- La interacción del sistema con los actores y qué cambios producen.

Una vez terminada la descripción se convoca a una revisión informal para determinar si: se han capturado como casos de uso todos los requisitos funcionales, la secuencia de acciones es completa y comprensible para cada caso de uso y si se identifica algún caso de uso que no proporcione valor, si es así se reconsiderara.

Actividad: Priorizar Casos de Uso

El propósito de esta actividad es determinar cuales son los casos de uso más importantes para desarrollar en las primeras iteraciones (es decir, análisis, diseño e implementación).

Formalización de la descripción de casos de uso

En algunas ocasiones los casos de uso pueden ser muy complejos, por eso puede llegar a ser necesaria una técnica de descripción más estructurada. La transición de estados puede ser tan compleja que resultaría casi imposible de mantener una descripción textual consistente, entonces se puede utilizar una técnica de modelado visual para la descripción de los casos de uso, los diagramas que se pueden utilizar son: diagramas de estado, diagramas de actividad, o diagramas de interacción.

Actividad: Prototipar la Interfaz de Usuario

Una vez realizado el modelo de casos de uso y la descripción del mismo se debe construir el prototipo de interfaz de usuario que permitirá llevar a cabo los casos de uso de manera eficiente. En primer lugar se intentara discernir que se necesita de las interfaces de usuario para habilitar los casos de uso para cada actor. Esto es el diseño lógico de la interfaz de usuario. Después se crea el diseño físico de la interfaz de usuario y se desarrolla prototipos que ilustren como pueden utilizar el sistema los usuarios para ejecutar los casos de uso.

Actividad: Estructurar el Modelo de Casos de Uso

Una vez realizado el modelo de casos de uso y su descripción se buscan comportamientos comunes para así reestructurar el modelo y que sea más fácil de entender. En la búsqueda de este tipo de comportamientos comunes y extensiones entre los casos de uso se encuentran:

- Relación de generalización: La generalización entre casos de uso es una clase de herencia, en la que las instancias de los casos de uso generalizados pueden ejecutar todo el comportamiento descrito en el caso de uso generalizador.
- Relación de extensión (<<extend>>): Esta relación modela la adición de una secuencia de acciones a un caso de uso. Una extensión se comporta como si fuera algo que se añade a la descripción original de un caso de uso.
- Relación de inclusión (<<include>>): Especifica que el caso de uso origen incorpora explícitamente comportamiento de otro caso de uso en la posición especificada por el origen.

Flujo de Trabajo: Análisis

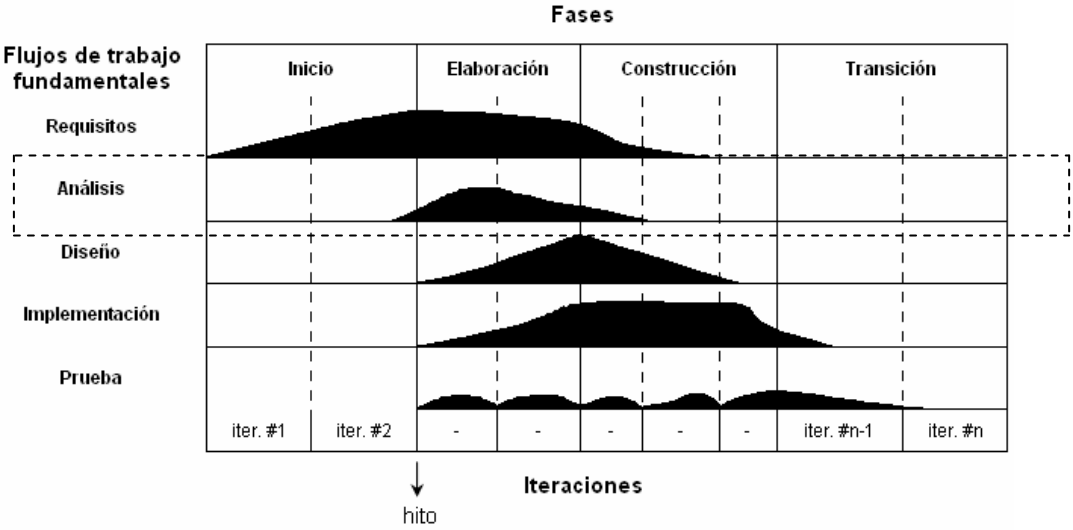
Durante el análisis, se analizan los requisitos que se describen en la captura de requisitos, definiéndolos y estructurándolos. El objetivo es conseguir una comprensión más precisa de los requisitos y una descripción de los mismos que sea fácil de mantener y que ayude a estructurar el sistema entero. Se utiliza un lenguaje más formal para apuntar detalles relativos a los requisitos del sistema, llamado "refinamiento de los requisitos".

Además, en el análisis se estructuran los requisitos de manera que facilite su comprensión, preparación, modificación, y, en general, su mantenimiento. Esta estructura es independiente de la estructura que se dio a los requisitos (basada en casos de uso). Sin embargo, existe una trazabilidad directa entre esas distintas estructuras, de forma que se puede hacer la traza de diferentes descripciones –en diferentes niveles de detalle– del mismo requisito y mantener su consistencia mutua con facilidad.

Modelo de casos de uso	Modelo de análisis
Descrito con el lenguaje del cliente.	Descrito con el lenguaje del desarrollador.
Vista externa del sistema.	Estructura por clases y paquetes estereotipados; proporciona la estructura a la vista interna.
Utilizado fundamentalmente como contrato entre el cliente y los desarrolladores sobre qué debería y qué no debería hacer el sistema.	Utilizado fundamentalmente por los desarrolladores para comprender como debería darse forma al sistema, es decir, como debería ser diseñado e implementado.
Puede contener redundancias, inconsistencias, etc., entre requisitos.	No debería contener redundancias, inconsistencias, etc., entre requisitos.
Captura la funcionalidad del sistema, incluida la funcionalidad significativa para la arquitectura.	Esboza cómo llevar a cabo la funcionalidad dentro del sistema, incluida la funcionalidad significativa para la arquitectura; sirve como una primera aproximación al diseño.
Define casos de uso que se analizarán con más profundidad en el modelo de análisis.	Define realizaciones de casos de uso, y cada una de ellas representa el análisis de un caso de uso del modelo de casos de uso.

El papel del análisis en el ciclo de vida del software

Las iteraciones iniciales de la elaboración se centran en el análisis. Eso contribuye a obtener una arquitectura más estable y sólida y facilita una comprensión en profundidad de los requisitos. Más adelante, al término de la fase de elaboración y durante la construcción, cuando la arquitectura es estable y se comprenden los requisitos, el énfasis pasa en cambio al diseño y la implementación.



Artefactos

Artefacto: Modelo de Análisis

El modelo de análisis se representa mediante un sistema de análisis que denota el paquete de alto nivel del modelo. La utilización de otros paquetes de análisis es por tanto una forma de organizar el modelo de análisis en partes más manejables que representan abstracciones de subsistemas y posiblemente capas completas del diseño del sistema. Dentro del modelo de análisis, los casos de uso se describen mediante clases de análisis y sus objetos. Esto se representa mediante colaboraciones llamadas “realización de caso de uso–análisis”.

Artefacto: Clases de Análisis

Una clase de análisis representa una abstracción de una o varias clases y/o subsistemas del diseño del sistema. Esta abstracción posee las siguientes características:

- Una clase de análisis se centra en el tratamiento de los requisitos funcionales y pospone los no funcionales, denominándolos requisitos especiales, hasta llegar a las actividades de diseño e implementación subsiguientes.

- Esto hace que una clase del análisis sea más evidente en el contexto del dominio del problema, más “conceptual”, a menudo de mayor granularidad que sus contrapartidas de diseño e implementación.
- Una clase de análisis raramente define u ofrece un interfaz en términos de operaciones y de sus signaturas. En cambio, su comportamiento se define mediante responsabilidades en un nivel más alto y menos formal. Una responsabilidad es una descripción textual de un conjunto cohesivo del comportamiento de una clase.
- Una clase de análisis define atributos, aunque esos atributos también son de un nivel bastante alto. Normalmente los tipos de los atributos son conceptuales y reconocibles en el dominio del problema. Además, los atributos identificados durante el análisis con frecuencia pasan a ser clases en el diseño y la implementación.
- Una clase de análisis participa en relaciones, aunque esas relaciones son más conceptuales que sus contrapartidas de diseño e implementación.
- Las clases del análisis siempre encajan en una de tres estereotipos básicos: limite, control o entidad. Cada estereotipo implica una semántica específica, lo cual constituye un método potente y consistente de identificar y describir las clases del análisis y contribuye a la creación de un modelo de objetos.

Clase limite

La clase limite se utiliza para modelar la interacción entre el sistema y sus actores. Esta interacción a menudo implica recibir (y presentar) información y peticiones (y hacia) los usuarios y los sistemas externos.

La clase limite modela las partes del sistema que dependen de sus actores, lo cual implica que clarifican y reúnen los requisitos en los límites del sistema. A menudo representa abstracciones de ventanas, formularios, paneles, interfaces de comunicaciones, interfaces de impresión, censores, terminales, y API (posiblemente no orientados a objetos). Aún así, la clase limite debería mantenerse a un nivel bastante alto y conceptual, es suficiente con que la clase describa lo que se obtiene con la interacción. Y no es necesario que describa cómo se ejecuta físicamente la interacción, ya que esto se considerará en las actividades de diseño e implementación.

Cada clase limite debería relacionarse con al menos un actor y viceversa.

Clase entidad

La clase entidad se utiliza para modelar información que posee una vida larga y que es a menudo persistente. La clase entidad modela la información y el comportamiento

asociado de algún fenómeno o concepto, como una persona, un objeto del mundo real, o un suceso del mundo real.

En la mayoría de los casos, las clases entidad se derivan directamente de una clase de entidad del negocio (o una clase del dominio) correspondiente, tomada del modelo de objetos del negocio (o del modelo del dominio). Sin embargo una diferencia fundamental entre clases entidad y clases de entidad del negocio es que las primeras representan objetos manejados por el sistema en consideración, mientras que las últimas representan objetos presentas en el negocio (y en el dominio del problema) en general. En consecuencia, las clases entidad reflejan la información de un modo que beneficia a los desarrolladores al diseñar e implementar el sistema, incluyendo su soporte de persistencia.

Las clases entidad suelen mostrar una estructura de datos lógica y contribuyen a comprender de qué información depende el sistema.

Clase de Control

Las clases de control representan coordinación, secuencia, transacciones, y control de otros objetos. Los aspectos dinámicos del sistema se modelan con clases de control, debido a que ellas manejan y coordinan las acciones y los flujos de control principales, y delegan trabajo a otros objetos (es decir, objetos limite y entidad).

Artefacto: Realización del Caso de Uso–Análisis

Una realización de caso de uso – análisis es una colaboración dentro del modelo de análisis que describe como se lleva a cabo y se ejecuta un caso de uso determinado en términos de las clases del análisis y de sus objetos del análisis. Una realización de caso de uso proporciona por tanto una traza directa hacia un caso de uso concreto del modelo de casos de uso.

Una realización de caso de uso posee una descripción textual del flujo de sucesos, diagramas de clases que muestran sus clases del análisis participantes, y diagramas de interacción que muestran la realización de un flujo o escenario particular del caso de uso en términos de interacción de objetos del análisis. Además, debido a que se describe una realización de caso de uso en términos de clases del análisis y de sus objetos, se centra de manera natural en los requisitos funcionales.

Diagramas de clases

Una clase de análisis y sus objetos normalmente participan en varias realizaciones de casos de uso, y algunas de las responsabilidades, atributos, y asociaciones de una clase concreta suelen ser sólo relevantes para una única realización de caso de uso.

Por tanto, es importante durante el análisis coordinar todos los requisitos sobre una clase y sus objetos que pueden tener diferentes casos de uso. Para hacerlo, se adjuntan diagramas de clases a las realizaciones de casos de uso, mostrando sus clases participantes y sus relaciones.

Flujo de sucesos–análisis

Los diagramas –especialmente los diagramas de colaboración– de una realización de caso de uso pueden ser difíciles de leer por sí mismos, de modo que pueden ser útil un texto adicional que los explique. Este texto debería escribirse en términos de objetos de control que interactúan para llevar a cabo el caso de uso. Sin embargo, el texto no debería mencionar ninguno de los atributos, responsabilidades, y asociaciones del objeto, debido a que cambian con bastante frecuencia y sería difícil mantenerlos.

Requisitos especiales

Los requisitos especiales son descripciones textuales que recogen todos los requisitos no funcionales sobre una realización de caso de uso. Algunos de estos requisitos ya se habían capturado de algún modo durante el flujo de trabajo de los requisitos, y sólo se cambian a una realización de caso de uso–análisis. Sin embargo, algunos de ellos pueden ser requisitos nuevos o derivados que se encuentran a medida que avanza el trabajo del análisis.

Artefacto: Descripción de la Arquitectura (vista del modelo de análisis)

La descripción de la arquitectura contiene una vista de la arquitectura del modelo de análisis, que muestra sus artefactos significativos para la arquitectura.

Los siguientes artefactos del modelo de análisis normalmente se consideran significativos para la arquitectura:

- La descomposición del modelo de análisis en paquetes de análisis y sus dependencias. Esta descomposición suele tener su efecto en los subsistemas de las capas superiores durante el diseño y la implementación y es por tanto relevante para la arquitectura en general.
- Las clases fundamentales del análisis como las clases entidad que encapsulan un fenómeno importante del dominio del problema; las clases limite que encapsulan interfaces de comunicación importantes y mecanismos de interfaz de usuario; las clases de control que encapsulan importantes secuencias con una amplia cobertura (es decir, aquéllas que coordinan realizaciones entre casos de uso

significativas). Puede ser suficiente con considerar significativa para la arquitectura una clase abstracta pero no sus subclases.

- Realizaciones de casos de uso que describen cierta funcionalidad importante y crítica; que implican muchas clases del análisis y por tanto tienen una cobertura amplia, posiblemente a lo largo de varios paquetes del análisis; o que se centran en un caso de uso importante que debe ser desarrollado al principio en el ciclo de vida del software, y por tanto es probable que se encuentre en la vida de la arquitectura del modelo de casos de uso.

Trabajadores

Trabajador: Arquitecto

Durante el flujo de trabajo del análisis, el arquitecto es responsable de la integridad del modelo de análisis, garantizando que éste sea correcto, consistente y legible como un todo. En sistemas grandes y complejos, estas responsabilidades pueden requerir más mantenimiento tras algunas iteraciones, y el trabajo que conllevan puede hacerse bastante rutinario. En esos casos el arquitecto puede delegar ese trabajo a otro trabajador, posiblemente un ingeniero de componentes de “alto nivel”. El arquitecto sigue siendo responsable de lo que es significativo para la arquitectura.

El arquitecto es también responsable de la arquitectura del modelo de análisis, es decir, de la existencia de sus partes significativas para la arquitectura.

Obsérvese que el arquitecto no es responsable del desarrollo y mantenimiento continuo de los diferentes artefactos del modelo de análisis

Trabajador: Ingeniero de Casos de Uso

Un ingeniero de casos de uso es responsable de la integridad de una o más realizaciones de caso de uso, garantizando que cumplen los requisitos que recaen sobre ellos. Una realización de caso de uso debe llevar a cabo correctamente el comportamiento de su correspondiente caso de uso del modelo de casos de uso, y sólo ese comportamiento.

Obsérvese que el ingeniero de casos de uso no es responsable de las clases del análisis ni de las relaciones que se usan en la realización del caso de uso.

El ingeniero de casos de uso también es responsable del diseño de las realizaciones de los casos de uso. Por tanto, el ingeniero de casos de uso es responsable tanto del análisis como del diseño del caso de uso, lo cual resulta en una transición suave.

Trabajador: Ingeniero de Componentes

El ingeniero de componentes define y mantiene las responsabilidades, atributos, relaciones, y requisitos especiales de una o varias clases del análisis, asegurándose de que cada clase del análisis cumple los requisitos que se esperan de ella de acuerdo a las realizaciones de caso de uso en las que participa.

El ingeniero de componentes también mantiene la integridad de uno o varios paquetes del análisis. Esto incluye garantizar que sus contenidos (por ejemplo, clases y sus relaciones) son correctos y que sus dependencias de otros paquetes del análisis son correctas y mínimas.

Suele ser apropiado dejar que el ingeniero de componentes responsable de un paquete del análisis lo sea también de las clases del análisis contenidas en él. Además, si existe una correspondencia directa entre un paquete del análisis y los subsistemas de diseño correspondientes, el ingeniero de componentes debería ser también responsable de esos subsistemas, para utilizar el conocimiento adquirido durante el análisis en el diseño y la implementación del paquete del análisis.

Flujo de trabajo

Los arquitectos comienzan la creación del modelo de análisis, identificando los paquetes de análisis principales, las clases entidad evidentes, y los requisitos comunes. Después, los ingenieros de caso de uso realizan cada caso de uso en términos de las clases de análisis participantes exponiendo los requisitos de comportamiento de cada clase. Los ingenieros de componentes especifican posteriormente estos requisitos y los integran dentro de cada clase creando responsabilidades, atributos y relaciones consistentes para cada clase.

Actividad: Análisis de la Arquitectura

El propósito del análisis de la arquitectura es esbozar el modelo de análisis y la arquitectura mediante la identificación de paquetes del análisis, clases de análisis evidentes, y requisitos especiales comunes.

Actividad: Analizar un Caso de Uso

- Identificar las clases de análisis cuyos objetos son necesarios para llevar a cabo el flujo de sucesos del caso de uso.
- Distribuir el comportamiento del caso de uso entre los objetos del análisis que interactúan.
- Capturar requisitos especiales sobre la realización del caso de uso. Refinar cada caso de uso en colaboraciones de clases de análisis.

Identificación de clases del análisis

En este paso, identificar las clases de control, entidad, y limite necesarias para realizar los casos de uso y esbozar sus nombres, responsabilidades, atributos y relaciones.

Los casos de uso descritos en los requisitos no siempre están suficientemente detallados como para poder identificar clases del análisis. Por tanto, para identificar las clases del análisis puede que se tengan que refinar las descripciones de los casos de uso.

Para identificar las clases del análisis:

- Identificar clases entidad mediante el estudio en detalle de la descripción del caso de uso y de cualquier modelo del dominio que se tenga, y después considerar que información debe utilizarse y manipularse en la realización del caso de uso.
- Identificar una clase limite central para cada actor humano, y dejar que esta clase represente la ventana principal de interfaz de usuario con el cual interactúa el actor.
- Identificar una clase limite primitiva para cada clase de entidad que se encontró anteriormente. Estas clases representan objetos lógicos con las cuales interactúa el actor (humano) en la interfaz de usuario durante el caso de uso.
- Identificar una clase limite central para cada actor que sea un sistema externo, y dejar que esta clase represente la interfaz de comunicación.
- Identificar una clase de control responsable del tratamiento del control y de la coordinación de la realización del caso de uso, y después refinar esta clase de control de acuerdo a los requisitos del caso de uso.

Utilizar un diagrama de clases para mostrar las relaciones que se utilizan en la realización del caso de uso.

Descripción de interacciones entre objetos del análisis

Se debe describir cómo interactúan los objetos del análisis. Esto se modela mediante diagramas de colaboración que contienen las instancias de actores participantes, los objetos del análisis, y sus enlaces. Si el caso de uso tiene flujos o subflujos diferenciados o distintos, suele ser útil crear un diagrama de colaboración para cada flujo. Esto contribuye a hacer más clara la realización del caso de uso, y también hace posible extraer diagramas de colaboración que representan interacciones generales y reutilizables.

En algunos casos es apropiado complementar el diagrama de colaboración con descripciones textuales, especialmente si el mismo caso de uso tiene muchos diagramas de colaboración que lo describen o si hay diagramas que representan flujos complejos. Estas descripciones textuales deberían recogerse en el artefacto flujo de sucesos–análisis de la realización del caso de uso.

Actividad: Analizar una Clase

Los objetivos de analizar una clase son:

- Identificar y mantener las responsabilidades de una clase de análisis, basadas en su papel en las realizaciones de caso de uso.
- Identificar y mantener los atributos y relaciones de la clase de análisis.
- Capturar requisitos especiales sobre la realización de la clase de análisis

Identificar responsabilidades

Las responsabilidades de una clase pueden recopilarse combinando todos los roles que cumple en diferentes realizaciones de caso de uso. Podemos identificar todas las realizaciones de caso de uso en las cuales participa la clase mediante el estudio de sus diagramas de clase y de interacción.

Identificar atributos

Un atributo especifica una propiedad de una clase, y normalmente es necesaria para las responsabilidades de su clase. Tener en cuenta:

- que el tipo del atributo debería ser conceptual en el análisis, y, si es posible, no debería verse restringido por el entorno de implementación.
- Al decidir el tipo de un atributo, se debe intentar reutilizar tipos ya existentes.
- Una determinada instancia de un atributo no puede compartirse por varios objetos del análisis. Si se necesita hacer esto, el atributo debe definirse en su propia clase.
- Si una clase del análisis es demasiado compleja por sus atributos, algunos de esos atributos podrían separarse en clases independientes.
- Los atributos de las clases entidad suelen ser bastante evidentes. Si una clase entidad tiene una traza con una clase del dominio, los atributos de esas clases son una entrada útil.
- Los atributos de las clases limite suelen representar elementos de información manipulados por los actores, tales como campos de texto etiquetados.
- Los atributos de las clases limite suelen representar propiedades de una interfaz de comunicación.

- Los atributos de las clases de control son poco frecuentes debido a su corto tiempo de vida. Sin embargo, las clases de control pueden poseer atributos que representan valores acumulados o calculados durante la realización de un caso de uso.
- A veces no son necesarios los atributos formales. En su lugar, puede ser suficiente con una sencilla explicación de una propiedad tratada por una clase de análisis, y pueden añadirse a la descripción de las responsabilidades de la clase.
- Si una clase tiene muchos atributos o atributos complejos, pueden mostrarlse en un diagrama de clases aparte, que sólo muestre la sección de atributos.

Identificación de asociaciones y agregaciones

Los objetos del análisis interactúan unos con otros mediante enlaces en los diagramas de colaboración. Estos enlaces suelen ser instancias de asociaciones entre sus correspondientes clases. Los enlaces pueden implicar la necesidad de referencias y agregaciones entre objetos.

Deberíamos minimizar el número de asociaciones entre clases. No son las relaciones del mundo real lo que deberíamos modelar como agregaciones o asociaciones, si no las relaciones que deben existir en respuesta a las demandas de las diferentes realizaciones de caso de uso.

Las agregaciones deberían utilizarse cuando los objetos representan:

- Conceptos que se contienen físicamente uno al otro, como un auto contiene al conductor.
- Conceptos que están compuestos uno de otro, como como un auto consta de un motor.
- Conceptos que forman una colección conceptual de objetos, como una familia que consta de un padre, una madre, y los hijos.

Captura de requisitos especiales

En este paso se recogen todos los requisitos de una clase de análisis, que se han identificado en la etapa de análisis pero que deberían tratarse en el diseño y en la implementación (requisitos no funcionales). Al llevar a cabo este paso, se debe estudiar los requisitos especiales de la realización del caso de uso, que pueden contener requisitos adicionales (no funcionales) sobre la clase de análisis.

Flujo de Trabajo: Diseño

En esta etapa se modela el sistema y se encuentra su forma (incluida su arquitectura) para que soporte todos los requisitos que se le suponen. Se debe conservar la estructura del sistema que se planteo en la etapa del análisis.

Los propósitos del diseño son:

- Comprender los aspectos relacionados con requisitos no funcionales, restricciones del lenguaje de programación, componentes reutilizables, sistemas operativos, concurrencia, tecnologías de interfaz de usuario, tecnologías de gestión transacciones, etc.
- Crear una entrada apropiada y un punto de partida para la implementación.
- Ser capaces de descomponer los trabajos de implementación en partes más manejables que puedan ser llevadas a cabo por diferentes equipos de desarrollo, teniendo en cuenta la posible concurrencia.
- Capturar las interfaces entre los subsistemas.
- Crear una abstracción de la implementación del sistema.

El papel del diseño en el ciclo de vida del software

El diseño es el centro de atención al final de la fase de elaboración y el comienzo de la construcción. Esto contribuye a una arquitectura estable y sólida y a crear un plano del modelo de implementación.

Artefactos

Artefacto: Modelo de Diseño

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso centrándose en como los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema.

El modelo de diseño se representa por un sistema de diseño que denota el subsistema de nivel más alto del modelo. La utilización de otro subsistema es, entonces, una forma de organización del modelo de diseño en porciones más manejables.

Los subsistemas de diseño y clases del diseño representan abstracciones del subsistema y componentes de la implementación del sistema.

En el modelo de diseño los casos de uso son realizados por las clases de diseño y sus objetos. Esto se representa por colaboraciones en el modelo de diseño y denota realización de caso de uso-diseño.

Artefacto: Clase del Diseño

Una clase del diseño es una abstracción de una clase o construcción similar en la implementación del sistema.

El lenguaje utilizado para especificar una clase del diseño es lo mismo que el lenguaje de programación. Consecuentemente, las operaciones, parámetros, atributos tipos y demás son especificados utilizando la sintaxis del lenguaje de programación elegido. También se deben incluir las asociaciones entre las clases.

Artefacto: Realización de Caso de Uso-Diseño

Una realización de caso de uso-diseño describe como se realiza un caso de uso específico, y como se ejecuta en términos de clases de diseño y sus objetos.

Una realización de caso de uso-diseño tiene una descripción de flujo de eventos textual, diagramas de clases que muestra sus clases de diseño participantes, y diagramas de interacción que muestra la realización de un flujo o escenario concreto de un caso de uso en términos de interacción entre objetos del diseño.

Diagramas de clases

Una clase de diseño y sus objetos, a menudo participan en más de una realización de casos de uso. También puede ocurrir que una operación o atributo de una clase sea específico de una realización de caso de uso. Se utilizan diagramas de clases conectados a una realización de caso de uso, mostrando sus clases participantes y sus relaciones.

Diagramas de interacción

Un caso de uso comienza cuando el actor envía algún tipo de mensaje al sistema. Algún objeto de diseño recibe el mensaje, dicho objeto envía un mensaje a otros objetos y de esta manera los objetos interactúan para llevar a cabo el caso de uso. El nombre del mensaje debería indicar una operación del objeto que recibe la invocación.

Para mostrar las interacciones entre objetos se utilizan el diagrama de secuencia y el de colaboración.

Flujo de sucesos-diseño

Una descripción textual del flujo de suceso puede ayudar a comprender los diagramas de interacción realizados. La descripción debería nombrar a los objetos que interactúan y no sus atributos, operaciones y asociaciones porque estas están sujetas a cambios.

Requisitos de implementación

Es una descripción textual, como los requisitos no funcionales sobre una realización de caso de uso, estos requisitos se pueden capturar en esta etapa o en etapas anteriores, pero se tratan en la etapa de implementación.

Artefacto: Subsistema de Diseño

Los subsistemas de diseño son una forma de organizar los artefactos del modelo de diseño en piezas más manejables. Un subsistema puede constar de clases del diseño, realizaciones de casos de uso, interfaces y otros subsistemas.

Artefacto: Descripción de la Arquitectura

La descripción de la arquitectura contiene una vista de la arquitectura del modelo de diseño, que muestra sus artefactos relevantes para la arquitectura.

Suelen considerarse significativos para la arquitectura los siguientes artefactos del modelo de diseño:

- Clases del diseño fundamentales, como clases que poseen una traza con clases del análisis significativas, clases activas, y clases del diseño que sean generales y centrales.
- Realizaciones de caso de uso-diseño que describan alguna funcionalidad importante y critica que debe desarrollarse pronto dentro del ciclo de vida del software, y además implique varias clases del diseño.

Artefacto: Modelo de Despliegue

El modelo de despliegue es un modelo de objetos que describe la distribución física del sistema en términos de cómo se distribuye la funcionalidad entre los nodos de computo (un procesador o un dispositivo hardware similar).

Trabajadores

Trabajador: Arquitecto

En el diseño, el arquitecto es responsable de la integridad de los modelos de diseño y de despliegue, garantizando que los modelos son correctos, consistentes y legibles en su totalidad.

Los modelos son correctos cuando realizan la funcionalidad descrita en el modelo de casos de uso, en los requisitos adicionales y en el modelo de análisis.

El arquitecto también es responsable de la arquitectura de los modelos de diseño y despliegue, es decir, de la existencia de sus partes significativas para la arquitectura, como se muestran en las vistas arquitectónicas de esos modelos.

Trabajador: Ingeniero de Casos de Uso

El ingeniero de casos de uso es responsable de la integridad de una o más realizaciones de casos de uso-diseño, y debe garantizar que cumplen los requisitos que se esperan de ellos.

Esto incluye hacer legibles y adecuadas para sus propósitos todas las descripciones textuales y todos los diagramas que describen la realización del caso de uso.

El ingeniero de casos de uso no es responsable de las clases, subsistemas e interfaces y relaciones de diseño que se utilizan en la realización del caso de uso.

Trabajador: Ingeniero de Componentes

El ingeniero de componentes define y mantiene las operaciones, métodos, atributos, relaciones y requisitos de implementación de una o más clases del diseño, garantizando que cada clase del diseño cumple los requisitos que se esperan de ella según las realizaciones de caso de uso en las que participa. El ingeniero de componentes también puede mantener la integridad de uno o más subsistemas. Suele ser adecuado que el ingeniero responsable de un subsistema sea también responsable de los elementos del modelo que este último contiene.

Flujo de trabajo

De acuerdo al comportamiento dinámico del diseño, los arquitectos comienzan la creación del modelo de diseño y despliegue esbozando los nodos del modelo de despliegue, los subsistemas principales y sus interfaces, las clases del diseño importantes como las activas, y los mecanismos genéricos de diseño del modelo de diseño. Después los ingenieros de casos de uso realizan los casos de uso en términos de clases y/o subsistemas del diseño participantes y sus interfaces. Los ingenieros de componentes especifican a continuación los requisitos, y los integran dentro de cada clase, bien mediante la creación de operaciones, atributos y relaciones consistentes sobre cada clase, o bien mediante la creación de operaciones consistentes en cada interfaz que proporcione el subsistema.

Actividad: Diseño de la Arquitectura

El objetivo del diseño de la arquitectura es esbozar los modelos de diseño y despliegue y su arquitectura mediante la identificación de los siguientes elementos:

- Nodos y sus configuraciones de red.
- Subsistemas y sus interfaces.
- Clases del diseño significativas para la arquitectura, como las clases activas.
- Mecanismos de diseño genéricos que tratan requisitos comunes, como los requisitos especiales sobre persistencia, distribución, rendimiento y demás.

Durante esta etapa se considerara la reutilización de partes de sistemas parecidos o productos software generales y se añadirán al modelo de diseño.

Identificación de subsistemas y de sus interfaces

Son un medio de organizar el modelo de diseño en piezas manejables. Se pueden identificar antes o durante la construcción del modelo de diseño.

No todos los subsistemas se desarrollan en esta etapa, algunos son productos reutilizados.

Identificación de clases del diseño relevantes para la arquitectura

La mayoría de las clases del diseño se identificarán al diseñar las clases dentro de la actividad del diseño de clases, y se refinarán de acuerdo a los resultados obtenidos en la actividad de diseño de casos de uso. Por este motivo se debería evitar identificar demasiadas clases en esta etapa.

Identificación de clases del diseño a partir de clases del análisis

A partir de las clases del análisis significativas para la arquitectura y las relaciones entre ellas, que encontramos en el análisis, se definen las clases del diseño y relaciones entre ellas.

Identificación de clases activas

El arquitecto también identifica las clases activas necesarias en el sistema considerando los requisitos de concurrencia del mismo. Por ejemplo: Los requisitos de rendimiento, tiempo de respuesta y disponibilidad que tienen los diferentes actores en su interacción con el sistema.

La distribución del sistema sobre los nodos

Otros requisitos como, el arranque y terminación del sistema, progresión, evitación del interbloqueo, evitación de la inanición, reconfiguración y la capacidad de los nodos.

Identificación de mecanismos genéricos de diseño

Se toman los requisitos especiales que se identificaron durante el análisis en las realizaciones de caso de uso-análisis y en las clases del análisis, y se decide como tratarlos, teniendo en cuenta las tecnologías de diseño e implementación disponibles. El resultado es un conjunto de mecanismos genéricos de diseño que pueden manifestarse como clases, colaboraciones o incluso subsistemas.

Los requisitos que deben tratarse suelen estar relacionados con aspectos como:

- Persistencia.
- Distribución transparente de objetos.
- Características de seguridad.
- Detección y recuperación de errores.
- Gestión de transacciones.

En algunos casos, el mecanismo no puede identificarse a priori, y se descubre en cambio a medida que se exploran las realizaciones de caso de uso y las clases de diseño.

Actividad: diseño de un caso de uso

Los objetivos de esta etapa son:

- Identificar las clases del diseño y/o los subsistemas cuyas instancias son necesarias para llevar a cabo el flujo de sucesos del caso de uso.
- Distribuir el comportamiento del caso de uso entre los objetos del diseño que interactúan y/o entre los subsistemas participantes.
- Definir los requisitos sobre las operaciones de las clases del diseño y/o sobre los subsistemas y sus interfaces.
- Capturar los requisitos de implementación del caso de uso.

Identificación de clases del diseño participantes

Para identificar las clases del diseño que participan en la realización de un caso de uso:

- Del estudio de las clases del análisis que participan de la realización del caso de uso-análisis se identifican las clases del diseño que tienen una traza con estas.

- Estudiar los requisitos especiales de la correspondiente realización de caso de uso-análisis. Identificar las clases del diseño que realizan esos requisitos especiales.

Descripción de las interacciones entre objetos del diseño

Se construyen, diagramas de secuencia que contienen las instancias de los actores, los objetos del diseño, y las transmisiones de mensajes entre éstos, que participan en el caso de uso. Si los casos de uso tienen subflujos se puede crear un diagrama de secuencia para estos.

Para crear un diagrama de secuencia, se comienza por el principio del flujo del caso de uso, y se sigue paso a paso, decidiendo qué objetos del diseño y qué interacciones de instancias de actores son necesarias para realizar cada paso. Se debe observar:

- La invocación de un caso de uso es un mensaje de una instancia de actor hacia un objeto del diseño.
- Cada clase del diseño que se identificó debería tener al menos una instancia en el diagrama de secuencia.
- Los mensajes que realizan el caso de uso se envían entre líneas de vida del objeto.
- La secuencia en el diagrama debería ser la principal preocupación, ya que la realización de caso de uso-diseño es la entrada principal para la implementación del caso de uso.
- El diagrama de secuencia debería tratar todas las relaciones del caso de uso que realiza.

Requisitos no funcionales implementación

En este paso, se incluyen en la realización del caso de uso todos los requisitos identificados durante el diseño que deberían tratarse en la implementación, como los requisitos no funcionales.

Actividad: Diseño de una Clase

El diseño de una clase incluye los siguientes aspectos:

- Sus operaciones.
- Sus atributos.
- Las relaciones en las que participa.
- Sus métodos (que realizan sus operaciones).
- Los estados impuestos.

- Sus dependencias con cualquier mecanismo de diseño genérico.
- Los requisitos relevantes a su implementación.
- La correcta realización de cualquier interfaz requerida.

Esbozar la clase del diseño

Cuando se dan como entrada una o varias clases del análisis, los métodos utilizados dependen del estereotipo utilizado en la clase de análisis:

- Diseñar clases limite es dependiente de la tecnología de interfaz especifica que se utilice.
- Diseñar clases entidad que representen información persistente a menudo implica el uso de tecnologías de bases de datos específicas.
- Al diseñar clases de control hay que tener en cuenta los siguientes aspectos: distribución, rendimiento y transacción.

Las clases del diseño identificadas en este paso deberán ser asignadas trazando dependencias a las correspondientes clases de análisis que son diseñadas.

Identificar operaciones

Se identifican las operaciones y se describen utilizando la sintaxis del lenguaje de programación. Algunas entradas importantes son:

- Las responsabilidades de cada clase, a menudo implican una o varias operaciones.
- Los requisitos especiales.
- Las interfaces que necesitan proporcionar.
- Las realizaciones de caso de uso diseño en las que la clase participa.

Identificar atributos

Se identifican atributos de las clases del diseño y se describen utilizando la sintaxis del lenguaje de programación. Un atributo especifica una propiedad de una clase. Cuando se identifican atributos se debe tener en cuenta lo siguiente:

- Los tipos de atributos están restringidos por el lenguaje de programación.
- Intentar reutilizar atributos existentes.
- Tener en cuenta si el atributo debe ser utilizado por varios objetos de la clase.
- Si una clase de diseño es muy complicada de entender sus atributos pueden ser separados en clases independientes.

Identificar asociaciones y agregaciones

Como los objetos del diseño interactúan unos con otros se deben modelar asociaciones entre sus clases. El número de relaciones debe estar minimizado y deben existir en respuesta a la realización de casos de uso. Como hay que tener en cuenta aspectos de rendimiento se deben modelar rutas de búsqueda óptimas a través de las asociaciones y agregaciones.

Cuando se definen asociaciones o agregaciones se debe tener en cuenta lo siguiente:

- Considerar las asociaciones y agregaciones involucrando la correspondiente clase análisis.
- Refinar la multiplicidad de las asociaciones, nombres de rol, clases de asociación, roles de ordenación, roles de cualificación, y asociaciones n-arias de acuerdo con el soporte del lenguaje de programación.
- Refinar la navegabilidad de las asociaciones.

Identificar las generalizaciones

Las generalizaciones (o herencia) se deben especificar usando la semántica del lenguaje de programación. En caso de que este no permita generalización se deberá usar asociación o agregación.

Describir los métodos

Los métodos especifican operaciones, se pueden describir usando lenguaje natural o pseudo código. No todos los métodos se describen en esta etapa la mayoría se describen en la etapa de implementación.

Describir estados

El estado de un objeto describe su comportamiento al recibir un mensaje, dicho estado se describe mediante un diagrama de estados.

Tratar Requisitos Especiales

En esta etapa se tratan los requisitos no funcionales que no se habían tratado hasta aquí, y se tienen en cuenta para la etapa de la implementación.

Actividad: Diseño de un Subsistema

- Garantizar que el subsistema es tan independiente como sea posible de otros subsistemas y/o de sus interfaces.
- Garantizar que el subsistema proporciona las interfaces correctas.

- Garantizar que el subsistema cumple su propósito de ofrecer una realización correcta de las operaciones.

Mantenimiento de las dependencias entre subsistemas

Se deben definir y mantener las dependencias de un subsistema con otros. En caso de que haya dependencias es mejor que un subsistema tenga dependencias con las interfaces de otros subsistemas. Debemos tratar de minimizar las dependencias entre subsistemas y/o interfaces.

Mantenimiento de interfaces proporcionadas por el subsistema

Las operaciones definidas por las interfaces que proporciona un subsistema deben soportar todos los roles que cumple el subsistema en las diferentes realizaciones de caso de uso.

Mantenimiento de los contenidos de los subsistemas

Un subsistema cumple sus objetivos cuando ofrece una realización correcta de las operaciones tal y como están descritas por las interfaces que proporciona.

Flujo de Trabajo: Implementación

Se parte del resultado del diseño y se implementa el sistema en término de componentes, es decir, ficheros de código fuente, scripts, ficheros de código binario, ejecutables y similares.

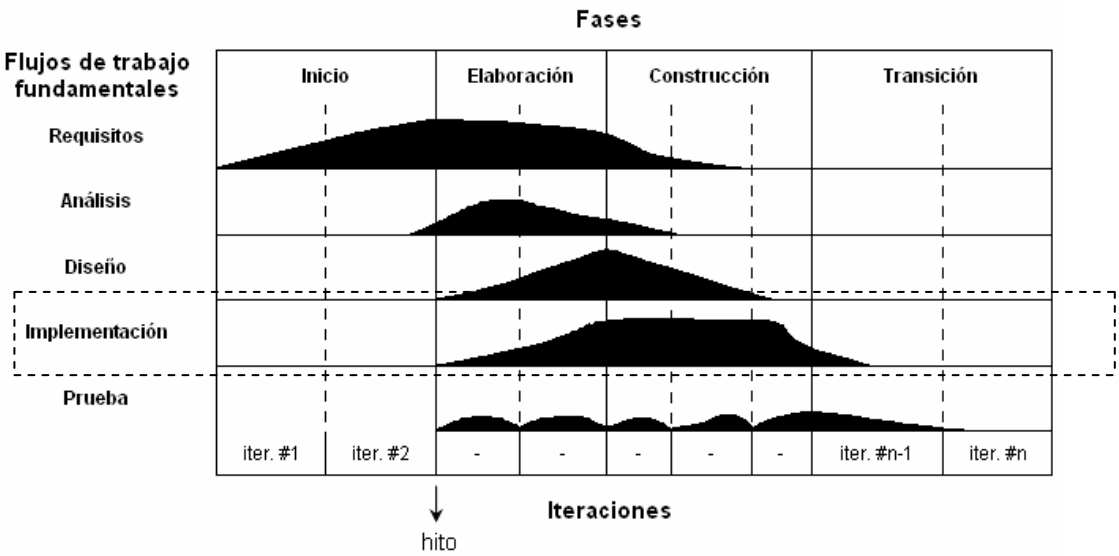
Afortunadamente, la mayor parte de la arquitectura del sistema es capturada durante el diseño, siendo el propósito principal de la implementación, desarrollar la arquitectura y el sistema como un todo. De forma más específica, los propósitos de la implementación son:

- Planificar las integraciones de sistema necesarias en cada iteración, siguiendo un enfoque incremental.
- Distribuir el sistema asignando componentes ejecutables a nodos en el diagrama de despliegue. Esto se basa fundamentalmente en las clases activas encontradas durante el diseño.
- Implementar las clases y subsistemas encontrados durante el diseño. En particular, las clases se implementan como componentes de fichero que contienen código fuente.

- Probar los componentes individualmente, y a continuación integrarlos compilándolos y enlazándolos en uno o más ejecutables, antes de ser enviados para ser integrados y llevar a cabo las comprobaciones de sistema.

El papel de la implementación en el ciclo de vida del software

La implementación es el centro durante las iteraciones de construcción, aunque también se lleva a cabo trabajo de implementación durante la fase de elaboración. Para crear la línea base ejecutable de la arquitectura, y durante la fase de transición,



para tratar defectos tardíos como los encontrados con distribuciones beta del sistema.

Ya que el modelo de implementación denota la implementación actual del sistema en términos de componentes y subsistemas de implementación, es natural mantener el modelo de implementación a lo largo de todo el ciclo de vida del software.

Artefactos

Artefacto: Modelo de Implementación

El modelo de implementación describe cómo los elementos del modelo de diseño, como las clases, se implementan en términos de componentes, como ficheros de código fuente, ejecutables, etc. El modelo de implementación describe también cómo se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en el lenguaje o lenguajes de programación utilizados, y cómo dependen los componentes unos de otros.

El modelo de implementación define una jerarquía y se representa con un sistema de implementación que denota el subsistema de nivel superior del modelo. El utilizar

otros subsistemas es por tanto una forma de organizar el modelo de implementación en trozos más manejables.

Artefacto: Componentes

Algunos estereotipos estándar de componentes son los siguientes:

- <<executable>> es un programa que puede ser ejecutado en un nodo.
- <<file>> es un fichero que contiene código fuente o datos.
- <<library>> es una librería estática o dinámica.
- <<table>> es una tabla de base de datos.
- <<document>> es un documento.

Durante la creación de componentes en un entorno de implementación particular estos estereotipos pueden ser modificados para reflejar el significado real de estos componentes. Los componentes tienen las siguientes características:

- Los componentes tienen relaciones con los elementos de modelo que implementan.
- Es normal que un componente implemente varios elementos, por ejemplo, varias clases; sin embargo, la forma exacta en que se crea esta traza depende de cómo van a ser estructurados y modularizados los ficheros de código fuente, dado el lenguaje de programación que se esté usando.
- Los componentes proporcionan las mismas interfaces que los elementos de modelo que implementan.
- Puede haber dependencias de compilación entre componentes, denotando qué componentes son necesarios para compilar un componente determinado.

Artefacto: Subsistema de Implementación

Los subsistemas de implementación proporcionan una forma de organizar los artefactos del modelo de implementación en trozos más manejables. Un subsistema puede estar formado por componentes, interfaces, y otros subsistemas. Además, un subsistema puede implementar las interfaces que representan la funcionalidad que exportan en forma de operaciones.

Es importante entender que un subsistema de implementación se manifiesta a través de un “mecanismo de empaquetamiento” concreto en un entorno de implementación determinado, tales como:

- Un paquete en java.
- Un paquete en una herramienta de modelo visual.

Los subsistemas de implementación están muy relacionados con los subsistemas de diseño en el modelo de diseño, “Artefacto: subsistema de diseño”. De hecho, los

subsistemas de implementación deberían seguir la traza uno a uno de sus subsistemas de diseño correspondientes.

Cualquier cambio en el modo en que los subsistemas proporcionan y usan interfaces, o cualquier cambio en las interfaces mismas, está descrito en el flujo de trabajo del diseño.

Artefacto: Interfaz

Las interfaces han sido descritas en detalle en el diseño. Sin embargo, las mismas interfaces pueden ser utilizadas en el modelo de implementación para especificar las operaciones implementadas por componentes y subsistemas de implementación. Además, como se mencionó anteriormente, los componentes y los subsistemas de implementación pueden tener “dependencias de uso” sobre interfaces.

Un componente que implementa (y por tanto proporciona) una interfaz ha de implementar correctamente todas las operaciones definidas por la interfaz. Un subsistema de implementación que proporciona una interfaz tiene también que contener la interfaz u otros subsistemas que proporcionen la interfaz.

Artefacto: Descripción de la Arquitectura (vista del modelo de implementación)

Los siguientes artefactos son considerados en el modelo de la implementación significativos arquitectónicamente:

- La descomposición del modelo de implementación en subsistemas, sus interfaces y las dependencias entre ellos. En general, esta descomposición es muy significativa para la arquitectura. Sin embargo, ya que los subsistemas de implementación siguen la traza de los subsistemas de diseño uno a uno y los subsistemas de diseño serán muy probablemente representados en la vista de la arquitectura del modelo de diseño, usualmente es innecesario representar los subsistemas de implementación en la vista de la arquitectura del modelo de implementación.
- Componentes clave, como los componentes que siguen la traza de las clases del diseño significativas arquitectónicamente, los componentes ejecutables y los componentes que son generales, centrales, que implementan mecanismos de diseño genéricos de los que dependen muchos otros componentes.

Artefacto: Plan de Integración de Construcción

Es importante construir el software incrementalmente en pasos manejables, de forma que cada paso dé lugar a pequeños problemas de integración o prueba. El resultado de cada paso es llamado “construcción”, que es una versión ejecutable del

sistema, usualmente una parte específica del sistema. Cada construcción es entonces sometida a pruebas de integración antes que se cree ninguna otra construcción. Para prepararse ante el fallo de una construcción, se lleva un control de versiones de forma que se pueda volver atrás a una construcción anterior. Los beneficios de este enfoque incremental son:

- Se puede crear una versión ejecutable del sistema muy pronto, en lugar de tener que esperar a una versión más completa.
- Es más fácil localizar defectos durante las pruebas de integración, porque sólo se añade o refina en una construcción existente una parte pequeña y manejable del sistema.
- Las pruebas de integración tienden a ser más minuciosas que las pruebas del sistema completo porque se centran en partes más pequeñas y más manejables.

Trabajadores

Trabajador: Arquitecto

Durante la fase de implementación, el arquitecto es responsable de la integridad del modelo de implementación y asegura que el modelo como un todo es correcto, completo y legible. El modelo es correcto cuando implementa la funcionalidad descrita en el modelo de diseño y en los requisitos adicionales (relevantes), sólo esta funcionalidad.

El arquitecto es responsable también de la arquitectura del modelo de implementación, es decir, de la existencia de sus partes significativas arquitectónicamente como se representó en la vista de la arquitectura del modelo de despliegue.

Por último, un resultado de la implementación es la asignación de componentes ejecutables a nodos. El arquitecto es responsable de la asignación, la cual se representa en la vista de la arquitectura del modelo de despliegue.

Trabajador: Ingeniero de Componentes

El ingeniero de componentes define y mantiene el código fuente de uno o varios componentes, garantizando que cada componente implementa la funcionalidad correcta. A menudo, también mantiene la integridad de uno o varios subsistemas de implementación. Ya que los subsistemas de implementación siguen la traza uno a uno de los subsistemas de diseño. Sin embargo, el ingeniero de componentes necesita garantizar que los contenidos (Ej.: los componentes) de los subsistemas de implementación son correctos, que sus dependencias con otros subsistemas o

interfaces son correctas y que implementan correctamente las interfaces que proporcionan.

Trabajador: Ingeniero de Sistemas

La integración del sistema está más allá del ámbito de cada ingeniero de componentes individuales. Por el contrario, su responsabilidad recae sobre el ingeniero de sistemas. Entre sus responsabilidades se incluye planificar la secuencia de construcciones necesarias en cada iteración y la integración de cada construcción cuando sus partes han sido implementadas. La planificación da lugar a un plan de integración de construcción.

Flujo de trabajo

El objetivo principal de la implementación es implementar el sistema. Este proceso es iniciado por el arquitecto esbozando los componentes clave en el modelo de implementación. A continuación, el ingeniero de sistemas planea las integraciones de sistemas necesarias en la presenta iteración como una secuencia de construcción. Para cada construcción, el ingeniero de sistema describe la funcionalidad que debería ser implementada y qué parte del modelo de implementación (es decir, subsistemas y componentes) se verán afectadas. Los requisitos sobre los subsistemas y componentes son entonces implementados por ingenieros de componentes. Los componentes resultantes son probados y pasados al ingeniero de sistemas para su integración.

Actividad: Implementación de la arquitectura

El propósito de la implementación de la arquitectura es esbozar el modelo de implementación y su arquitectura mediante:

- La identificación de componentes significativos arquitectónicamente, tales como componentes ejecutables.
- La asignación de componentes a nodos en las configuraciones de redes relevantes.

Recordemos que durante el diseño de la arquitectura se esbozan los subsistemas de diseño, sus contenidos e interfaces. Durante la implementación utilizamos subsistemas de implementación que siguen la traza uno a uno a estos subsistemas de diseño y proporcionan las mismas interfaces.

Identificación de componentes ejecutables y asignación de estos a nodos

Para identificar los componentes ejecutables que puedan ser desplegados sobre los nodos, consideramos las clases activas encontradas durante el diseño y asignamos un componente ejecutable por cada clase activa, denotando así un proceso pesado. Esto podría incluir la identificación de otros componentes de fichero o de código binario necesarios para crear los componentes ejecutables, aunque esto es secundario.

Actividad: Integrar el Sistema

Los objetivos de integrar el sistema son:

- Crear un plan de integración de construcciones que describan las construcciones necesarias en una iteración y los requisitos de cada construcción.
- Integrar cada construcción antes de que sea sometida a pruebas de integración.

Integración de una construcción

Si la construcción ha sido planificada cuidadosamente, debería ser fácil integrar la construcción. Esto se hace recopilando las versiones correctas de los subsistemas de implementación y de los componentes, compilándolos y enlazándolos para generar una construcción. La construcción resultante se somete a pruebas de integración, y a pruebas de sistema si pasa las pruebas de integración y es la última construcción dentro de una iteración.

Actividad: Implementar un Subsistema

El propósito de implementar un subsistema es el de asegurar que un subsistema cumple su papel en cada construcción, tal y como se especifica en el plan de integración de la construcción. Esto quiere decir que se asegura que los requisitos implementados en la construcción y aquellos que afecten al subsistema son implementados correctamente por componentes o por otros subsistemas dentro del subsistema.

Actividad: Implementar una Clase

El propósito de la implementación de una clase es implementar una clase de diseño en un componente fichero. Esto incluye lo siguiente:

- Esbozo de un componente fichero que contendrá el código fuente.
- Generación de código fuente a partir de la clase de diseño y de las relaciones en que participa.

- Implementación de las operaciones de la clase de diseño en forma de métodos.
- Comprobación de que el componente proporciona las mismas interfaces que la clase de diseño.

Esbozo de los componentes fichero

El código fuente que implementa una clase de diseño reside en un componente fichero. Por tanto, se esboza el componente fichero y se considera su ámbito. Es normal implementar varias clases de diseño en un mismo componente fichero.

Generación de código a partir de una clase de diseño

A lo largo del diseño, muchos de los detalles relacionados con la clase de diseño y con sus relaciones son descritos utilizando la sintaxis del lenguaje de programación elegido, lo que hace que la generación de partes del código fuente que implementan la clase sea muy fácil. En particular, esto se cumple para las operaciones y atributos de las clases, así como para las relaciones en las que la clase participa.

Implementación de operaciones

Cada operación definida por la clase de diseño ha de ser implementada, a no ser que sea “virtual” (o “abstracta”) y ésta sea implementada por descendientes (como subtipos) de la clase. Se utiliza el término *métodos* para denotar la implementación de operaciones.

La implementación de una operación incluye la elección de un algoritmo y unas estructuras de datos apropiadas, y la codificación de las acciones requeridas por el algoritmo.

Actividad: Realizar Prueba de Unidad

El propósito de realizar la prueba de unidad es probar los componentes implementados como unidades individuales. Se llevan a cabo los siguientes tipos de pruebas de unidad:

- La *prueba de especificación*, o “prueba de caja negra”, que verifica el comportamiento de la unidad observable externamente.
- La *prueba de estructura,* o “prueba de caja blanca”, que verifica la implementación interna de la unidad.

Además, han de ser realizadas las pruebas de integración y sistema para asegurar que los diversos componentes se comportan correctamente cuando se integran.

Realización de pruebas de especificación (prueba de caja negra)

La prueba de especificación se realiza para verificar el comportamiento del componente sin tener en cuenta cómo se implementa dicho comportamiento en el componente. Las pruebas de especificación, por tanto, tienen en cuenta la salida que el componente devolverá cuando se le dá una determinada entrada en un determinado estado. El número de combinaciones de posibles entradas, estados iniciales y salidas es a menudo considerable, lo que hace impracticable probar todas las combinaciones una a una. En su lugar, el número de entradas, salidas y estados se divide en clases de equivalencia. Una clase de equivalencia es un conjunto de valores de entrada, estado o salida para los que se supone que un objeto se comporta de forma similar. Probando un componente para cada combinación de las clases de equivalencia de entradas, salidas y estados iniciales es posible obtener casi la misma cobertura "efectiva" de prueba que si se prueban individualmente cada una de las combinaciones de valores, pero con un esfuerzo considerablemente menor.

Realización de pruebas de estructura (prueba de caja blanca)

Las pruebas de estructura se realizan para verificar que un componente funciona internamente como se quería. El ingeniero de componentes debería asegurarse de probar todo el código durante las pruebas de estructuras, lo que quiere decir que cada sentencia ha de ser ejecutada al menos una vez. El ingeniero de componentes debería también asegurarse de probar los caminos más importantes en el código. Entre estos caminos tenemos los que son seguidos más comúnmente, los caminos críticos, los caminos menos conocidos de los algoritmos y otros caminos asociados con riesgos altos.