Construcción de un Modelo Conceptual para Gramáticas Formales y Máquinas Abstractas con UML usando Rational

Torres Díaz, Melisa Daniela

Universidad Tecnológica Nacional, Facultad Regional Córdoba

Abstract

El presente es un proyecto, llevado a cabo en la Universidad Tecnológica Nacional, Facultad Regional Córdoba durante el año 2007 en carácter de trabajo de investigación. Este trabajo tiene como objetivo la comparación de la metodología y herramientas para la especificación de requerimientos con el fin de determinar un esquema conceptual sobre un dominio de aplicación determinado: las Gramáticas Formales y las Máquinas Abstractas. La metodología aplicada para el desarrollo de este trabajo es Rational Unified Process (RUP/UML), con la utilización de la herramienta de modelado Rational.

Palabras Clave

Máquinas Abstractas. Autómatas. Gramáticas Formales. Isomorfismo. Rational Unified Process (RUP). Unified Modeling Languaje (Lenguaje Unificado de Desarrollo, UML). Casos de Uso. Diagramas de Casos de Uso, Clases, Objetos, Componentes, de Despliegue, de Transición de Estados, de Interacción. Trazabilidad entre casos de uso. Workflows. Modelo Objeto Dominio del Problema (MODP)

Introducción

El objetivo de este proyecto, consiste en realizar la construcción de un modelo conceptual para Gramáticas Formales y Máquinas Abstractas, para la determinación de una técnica de especificación de requerimientos, específicamente de un esquema conceptual, realizado sobre éste dominio de aplicación definido.

El criterio de elección del dominio de Máquinas Abstractas y Gramáticas Formales, tiene por fundamentación adicional, no solo el de ser un dominio al que se puede reducir la ambigüedad semántica al estar perfectamente definido, sino que también presenta la ventaja de ser modelos isomorfos entre sí, el cuál permitirá evaluar en que medida los metamodelos resultantes de la aplicación de la metodología continúan siéndolo.

Para el desarrollo de este proyecto fue necesaria primeramente la descripción exhaustiva de las Gramáticas Formales y las Máquinas Abstractas que serán la entrada del modelo desarrollado con RUP/UML, con la ayuda de la poderosa herramienta Rational.

Luego, se procedió a realizar un profundo estudio sobre la metodología y la herramienta a utilizar para el modelado. Por último, y teniendo bien en claro todos los conceptos a manejar, se procedió a modelar las Gramáticas Formales y Máquinas Abstractas.

Elementos del Trabajo y metodología

Para poder realizar un estudio comparativo de metodologías y herramientas existentes, para la determinación de una técnica de especificación de requerimientos, específicamente de un esquema conceptual, realizado sobre un dominio de aplicación definido, como lo son las Gramáticas Formales y las Máquinas Abstractas, fue necesario primeramente describir exhaustivamente el objeto bajo estudio, ya que son la base y única entrada del modelo realizado aplicando la metodología RUP/UML con la utilización de la herramienta Rational.

Puesto en claro qué es lo que se quiere desarrollar, se prosiguió con el estudio de la metodología, haciendo hincapié en cómo surgió, las características esenciales de la misma, las fases dentro de un ciclo de vida de un proyecto, los pilares del RUP en el desarrollo del software; pero por otro lado también se explicó qué es el UML, y sus características y notaciones que se utilizan para modelar elementos, relaciones y diagramas. Por otro lado, se hizo una breve reseña de Rational, herramienta utilizada para desarrollar el modelo resultante de la aplicación de la metodología ya mencionada.

Una vez que se estudió la metodología y la herramienta a utilizar, se explicó detalladamente cómo se procedió para modelar las Gramáticas Formales y Máquinas Abstractas.

Historia del RUP

La Figura 1 ilustra la historia de RUP. El antecedente más importante se ubica en 1967 con la Metodología Ericsson (Ericsson Approach) elaborada por Ivar Jacobson, una aproximación de desarrollo basada en componentes, que introdujo el concepto de Caso de Uso. Entre los años de 1987 a 1995 Jacobson fundó la compañía Objectory AB y lanza el proceso de desarrollo Objectory (abreviación de Object Factory).

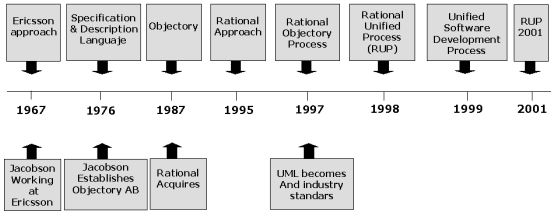


Figura 1: Historia de RUP

Posteriormente en 1995 Rational Software Corporation adquiere Objectory AB y entre 1995 y 1997 se desarrolla Rational Objectory Process (ROP) a partir de Objectory 3.8 y del Enfoque Rational (Rational Approach) adoptando UML como lenguaje de modelado.

Desde ese entonces y a la cabeza de Grady Booch, Ivar Jacobson y James Rumbaugh, Rational Software desarrolló e incorporó diversos elementos para expandir ROP, destacándose especialmente el flujo de trabajo conocido como modelado del negocio. En junio del 1998 se lanza Rational Unified Process.

Características Esenciales

El Proceso Unificado es un proceso de desarrollo de software. Un proceso de desarrollo de software es un conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema de software. Es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas de software, para diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de aptitud y diferentes tamaños de proyecto.

El Proceso Unificado utiliza el Lenguaje Unificado de Modelado (Unified Modeling Languaje, UML) para preparar todos los esquemas de un sistema de software. De hecho, UML es una parte esencial del Proceso Unificado (sus desarrollos fueron paralelos).

Los aspectos definitorios del Proceso Unificado es que es dirigido por casos de uso, centrado en la arquitectura, e iterativo e incremental.

Proceso dirigido por Casos de Uso

Los Casos de Uso son una técnica de captura de requisitos que fuerza a pensar en términos de importancia para el usuario y no sólo en términos de funciones que seria bueno contemplar. Se define un Caso de Uso como un fragmento de funcionalidad del sistema que proporciona al usuario un valor añadido. Los Casos de Uso representan los requisitos funcionales del sistema.

En RUP los Casos de Uso no son sólo una herramienta para especificar los requisitos del sistema. También guían su diseño, implementación y prueba. Los Casos de Uso constituyen un elemento integrador y una guía del trabajo.

Los Casos de Uso no sólo inician el proceso de desarrollo sino que proporcionan un hilo conductor, permitiendo establecer trazabilidad entre los artefactos que son generados en las diferentes actividades del proceso de desarrollo (o también llamados workflows). Dirigido por casos de uso quiere decir que el proceso de desarrollo sigue un hilo: avanza a través de una serie de flujos de trabajo que parten de los casos de uso.

Como se muestra en la Figura 2, basándose en los Casos de Uso se crean los modelos de análisis y diseño, luego la implementación que los lleva a cabo, y se verifica que efectivamente el producto implemente adecuadamente cada Caso de Uso. Todos los modelos deben estar sincronizados con el modelo de Casos de Uso.

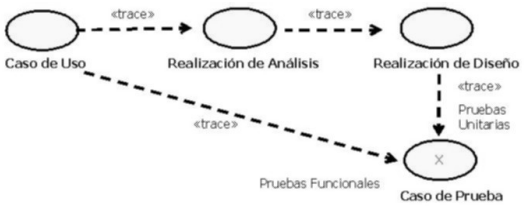


Figura 2: Trazabilidad a partir de los Casos de Uso

Proceso centrado en el Arquitectura

La arquitectura de un sistema es la organización o estructura de sus partes más relevantes, lo que permite tener una visión común entre todos los involucrados (desarrolladores y usuarios) y una perspectiva clara del sistema completo, necesaria para controlar el desarrollo.

El concepto de arquitectura de software incluye los aspectos estáticos y dinámicos más significativos del sistema. La arquitectura surge de las necesidades de la empresa, como las perciben los usuarios y los inversores, y se refleja en los casos de uso. Es influenciada por otros factores: plataforma en la que tiene que funcionar el software, los bloques de construcción reutilizables de que se dispone, consideraciones de implantación, sistemas heredados, y requisitos no funcionales.

Los arquitectos deben trabajar sobre la comprensión general de las funciones claves: los casos de uso claves del sistema. Un arquitecto:

- Crea un esquema en borrador de la

arquitectura, comenzando por la arte que no es especifica de los casos de uso.

- Trabaja con un subconjunto de los casos de uso especificados, con aquellos que representen as funciones claves del sistema en desarrollo. Cada caso de uso seleccionado se especifica en detalle y se realiza en términos de subsistemas, clases y componentes.
- A medida que los casos de uso se especifican y maduran, se descubre más de la arquitectura. Esto, a su vez, lleva a la maduración de más casos de uso.

Este proceso continúa hasta que se considere que la arquitectura es estable.

La arquitectura se describe a través de vistas:

Figura 3: Modelado de la arquitectura de un sistema

La vista de casos de uso de un sistema comprende los casos de uso que describen el comportamiento del sistema tal y como es percibido por los usuarios finales, analistas, y encargados de prueba. Con UML, los aspectos estáticos de esta vista se capturan en los diagramas de casos de uso; los aspectos dinámicos de esta vista se capturan en los diagramas de interacción, de transición de estados y actividades.

La vista de vista de diseño de un sistema comprende las clases, interfaces y colaboraciones que forman el vocabulario del problema y su solución. Esta vista soporta principalmente los requisitos funcionales del sistema, entendiendo por ello los servicios que el sistema debería proporcionar a sus usuarios finales. Con UML, los aspectos estáticos de esta vista se capturan en los diagramas de clases y de objetos; y los aspectos dinámicos se capturan en los diagramas de interacción, diagramas de transición de estados y de actividades.

La vista de procesos de un sistema comprende los hilos y procesos que forman los mecanismos de sincronización y concurrencia del sistema. Esta vista cubre principalmente el funcionamiento, capacidad de crecimiento y rendimiento del sistema. Con UML, los aspectos estáticos y dinámicos de esta vista se capturan en el mismo tipo de diagramas que la vista de diseño, pero con énfasis en las clases activas que representan estos hilos y procesos.

La vista de implementación de un sistema comprende los componentes y archivos que se utilizan para ensamblar y hacer disponible.

Proceso iterativo e incremental

El equilibrio correcto entre los Casos de Uso y la arquitectura es algo muy parecido al equilibrio de la forma y la función en el desarrollo del producto, lo cual se consigue con el tiempo. Para esto, la estrategia que se propone en RUP es tener un proceso iterativo e incremental en donde el trabajo se divide en partes más pequeñas o mini proyectos. Permitiendo que el equilibrio entre Casos de Uso y arquitectura se vaya logrando durante cada mini proyecto, así durante todo el proceso de desarrollo. Cada mini proyecto se puede ver como una iteración (un recorrido más o menos completo a lo largo de todos los flujos de trabajo fundamentales) del cual se obtiene un incremento que produce un crecimiento en el producto.

Una iteración puede realizarse por medio de una cascada como se muestra en la Figura 5. Se pasa por los flujos fundamentales (Requisitos, Análisis, Diseño, Implementación y Pruebas), también existe una planificación de la iteración, un análisis de la iteración y algunas actividades específicas de la iteración. Al finalizar se realiza una integración de los resultados con lo obtenido de las iteraciones anteriores.

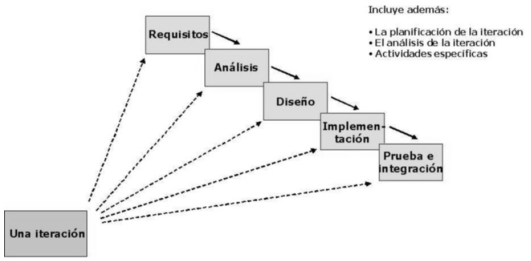


Figura 5: Una iteración RUP

El proceso iterativo e incremental consta de una secuencia de iteraciones. Cada iteración aborda una parte de la funcionalidad total, pasando por todos los flujos de trabajo relevantes y refinando la arquitectura. En cada iteración, los desarrolladores identifican y especifican los casos de uso más relevantes, crean un diseño utilizando la arquitectura seleccionada como guía, implementan el diseño mediante componentes, y verifican que los componentes satisfagan los casos de uso. Si una iteración cumple con sus objetivos, el

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

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. La eliminación de una de las tres ideas reduciría drásticamente el valor del Proceso Unificado.

Fases dentro de un ciclo de vida

Cada ciclo se desarrolla a lo largo del tiempo. Cada fase termina con un hito, que determina la disponibilidad de un conjunto de artefactos; es decir, ciertos modelos o documentos han sido desarrollados hasta alcanzar un estado determinado.

Los hitos también permiten a la dirección, y a los mismos desarrolladores, controlar el progreso del trabajo según pasa por esos cuatro puntos clave.

La figura 6 muestra en la columna izquierda los flujos de trabajo (requisitos, análisis, diseño, implementación y prueba). Las curvas son una aproximación (no deben considerarse muy literalmente) de hasta dónde se llevan a cabo los flujos de trabajo en cada fase. Una iteración típica pasa por los cinco flujos de trabajo (o workflow), como se muestra en la siguiente figura 6:

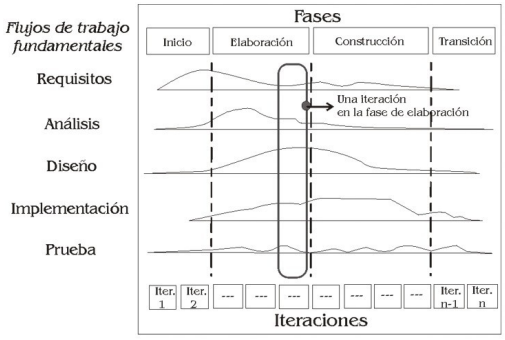


Figura 6: Esfuerzo en actividades según fase del proyecto

Durante la fase de inicio, se desarrolla una descripción del producto final a partir de una buena idea y se presenta el análisis del negocio para el producto.

Durante la fase de elaboración, se especifica en detalle la mayoría de los casos de uso del producto y se diseña la arquitectura del sistema. La relación entre la arquitectura del sistema 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 que permite que el esqueleto haga movimientos básicos). El sistema es el cuerpo entero con esqueleto, piel, y músculos.

La arquitectura se expresa en forma de vistas de todos los modelos del sistema, los cuales juntos representan al sistema entero. Esto implica que hay varias vistas arquitectónicas del modelo de casos de uso, del modelo de análisis, del modelo de diseño, del modelo de implementación y modelo de despliegue. Durante la fase de desarrollo, se realizan los casos de uso más críticos que se identificaron en la fase de comienzo. El resultado de esta fase es una línea base de la arquitectura.

Durante la fase de construcción se crea el producto (se añaden músculos (software terminado) al esqueleto (la arquitectura)). 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 se descubrirán y corregirán en la fase de transición.

La fase de transición cubre el periodo 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 defectos y deficiencias. La fase de transición conlleva actividades como la fabricación, formación del cliente, el proporcionar una línea de ayuda y asistencia, y la corrección de los defectos que se encuentren tras la entrega.

Pilares del RUP – 4P

El resultado final del proyecto software es un producto que toma la forma durante su desarrollo gracias a la intervención de muchos tipos de personas.

Los términos personas, proyecto, producto, proceso y herramientas serán definidos a continuación:

*Personas: los principales autores de un proyecto software son los arquitectos, desarrolladores, ingenieros de prueba, y el personal de gestión que les da soporte, además de los usuarios, clientes, etc. Las personas son realmente seres humanos a diferencia del término abstracto trabajadores (que mas adelante será definido).

*Proyecto: elemento organizativo a través del cual se gestiona el desarrollo de software. El resultado de un proyecto es una versión del producto.

*Producto: artefactos que se crean durante la vida del proyecto, como los modelos, código fuente, ejecutables, y documentación.
Proceso: es una definición del conjunto completo de actividades necesarias para transformar los requisitos del usuario en un producto. Un proceso es una plantilla para crear proyectos.

*Herramientas: software que se utiliza para automatizar las actividades definidas en el proceso.

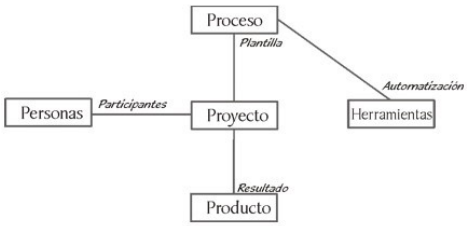


Figura 7: Las cuarto "P" en el desarrollo de software

Los procesos de desarrollo afectan a las personas

El modo en que se organiza y gestiona un proyecto software afecta a las personas implicadas en él. Conceptos como viabilidad, la gestión de riesgo, la organización de los equipos, la planificación del proyecto y la facilidad de comprensión del proyecto tienen un papel importante:

-Viabilidad del proyecto: la mayoría de la gente no disfruta trabajar en un proyecto que parece imposible de realizar. Los proyectos que no son viables suelen terminar en una fase temprana, aliviando así problemas de moral.

-Gestión del riesgo: de igual forma, cuando la gente siente que los riesgos no han sido analizados y reducidos, se sienten incómodos. La exploración en las primeras etapas atenúa este problema.

-Estructura de los equipos: es conveniente que los grupos de trabajo sean pequeños, ya que se trabaja de una forma más cómoda.

-Planificación del proyecto: las técnicas que se utilizan en las fases de inicio y de elaboración permiten a los desarrolladores tener una buena noción de cuál debería ser el resultado del proyecto.

-Facilidad de comprensión del proyecto: a la gente le gusta saber qué es lo que se está haciendo; quieren tener una comprensión global. La descripción de la arquitectura proporciona una visión general para cualquiera que se encuentre implicado en el proyecto.

Debido a que las personas son las que ejecutan las actividades clave del desarrollo de software, es necesario un proceso de desarrollo que esté soportado por herramientas y un Lenguaje Unificado de Modelado, para hacer que las personas sean más eficaces. Ese proceso permitirá a los desarrolladores construir un mejor software en términos de tiempo de salida al mercado, calidad y costes. Les permite especificar los requisitos que mejor se ajusten a las necesidades de los usuarios; como así también elegir una arquitectura que permita construir los sistemas de forma económica y puntual.

La gente llega a ocupar muchos puestos diferentes en una organización de desarrollo de software. Se llama trabajador a los puestos a los cuales pueden asignar a personas, y los cuales esas personas aceptan; es definitiva es un papel que un individuo puede desempeñar en el desarrollo de software. También es importante dar la definición de rol. Un rol es un papel que cumple un trabajador. Un trabajador es responsable de un conjunto completo de actividades. Al mismo tiempo, si tiene que hacer su trabajo, las herramientas que emplean deben ser las adecuadas. Las herramientas no sólo deben ayudar a los trabajadores a llevar a cabo sus propias actividades, sino también deben aislarles de la información que no le es relevante. Para lograr estos objetivos, el Proceso Unificado describe formalmente los puestos (es decir, los trabajadores), que las personas pueden asumir en el proceso. Un trabajador también puede representar a un conjunto de personas que trabajan juntas. Cada trabajador tiene un conjunto de responsabilidades y lleva a cabo un conjunto de actividades en el desarrollo de software.

El producto es más que código

Un proyecto de desarrollo da como resultado una nueva versión de un producto. En el contexto del Proceso Unificado, el producto que se obtiene es un sistema de software.

Un sistema es todos los artefactos que se necesitan para representarlos en una forma comprensible pro máquinas u hombres, para las máquinas (herramientas, compiladores, etc.), los trabajadores (directores, arquitectos, desarrolladores) y los interesados (inversores, usuarios, jefes de proyecto).

Un 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 mas utilizado en el Proceso Unificado es el modelo. La construcción de un sistema es un proceso de construcción de

modelos, utilizando distintos modelos para describir todas las perspectivas diferentes del sistema.
Un modelo es una abstracción del sistema, especificando el sistema modelado desde cierto punto de vista y en un determinado nivel de abstracción. Un punto de vista es, por ejemplo, una vista de especificación o una vista de diseño del sistema.

El proceso: una plantilla

En el Proceso Unificado, proceso hace referencia a un contexto que sirve como plantilla que puede reutilizarse para crear instancias de ella. Es comparable a una clase, que puede utilizarse para crear objetos en el paradigma orientado a objetos. Instancia del proceso es un sinónimo de proyecto.
Un proceso de desarrollo de software es un conjunto completo de actividades necesarias para convertir los requisitos de usuario en un conjunto consistente de artefactos que conforman un producto de software, y para convertir los cambios sobre esos requisitos en un nuevo conjunto consistente de artefactos.
Un proceso es una definición de un conjunto de actividades, no su ejecución.

Las herramientas son esenciales en el proceso

Las herramientas soportan los procesos de desarrollo de software modernos.
El proceso se ve influido fuertemente por las herramientas. Las herramientas son buenas para automatizar procesos repetitivos, mantener las cosas estructuradas, gestionar grandes cantidades de información y para guiarnos a lo largo de un camino de desarrollo concreto.
Las herramientas se desarrollan para automatizar actividades, de manera completa o parcial, para incrementar la productividad y la calidad, y para reducir el tiempo de desarrollo.

¿Qué es UML?

El Proceso Unificado utiliza el Lenguaje Unificado de Modelado (UML) para preparar todos los esquemas de un sistema de software. De hecho, UML es una parte esencial del Proceso Unificado (sus desarrollos fueron paralelos). A continuación, se muestra un cuadro del bloque de construcción de UML:

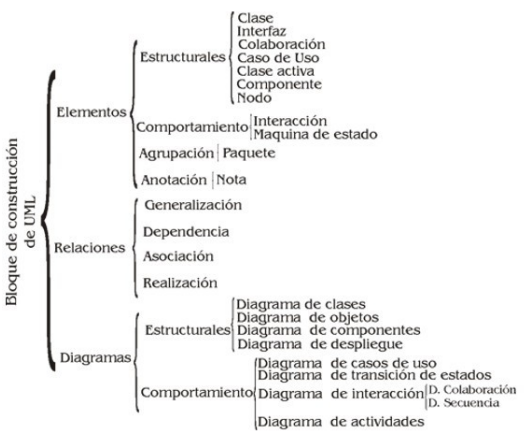


Figura 8: Bloques de construcción de UML

Estructurales

Son los nombres de los modelos de UML. En su mayoría son las partes estáticas de un modelo, y representan cosas que son conceptuales o materiales.
* Clase: es una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica. Una clase implementa una o más interfaces.
* Interface: es una colección de operaciones que especifican un servicio de una clase o componente. Describe un comportamiento visible externamente de ese elemento. Puede representar el comportamiento completo de una clase o componente o sólo una parte de ese comportamiento.
* 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. Por lo tanto, las colaboraciones tienen una dimensión tanto estructural como de comportamiento. Una clase dada puede participar en varias colaboraciones. Estas colaboraciones representan la implementación de patrones que forman un sistema.
* 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. Una clase activa es igual que una clase, excepto en que sus objetos representan elementos cuyo comportamiento es concurrente con otros elementos.
* Componente: Es una parte física y reemplazable de un sistema que conforma con un conjunto de interfaces y proporciona la implementación de dicho conjunto. En un sistema, se podrán encontrar diferentes tipos de componentes de despliegue (tales como componentes de Java), así como componentes que sean artefactos del proceso de desarrollo (tales como archivos de código fuente). Un componente representa típicamente el empaquetamiento físico de diferentes elementos lógicos, como clases, interfaces y colaboraciones.
* Nodo: es un elemento físico que existe en tiempo de ejecución y representa un recurso computacional, que por lo general dispone de algo de memoria y con frecuencia, capacidad de procesamiento. Un conjunto de

componentes puede residir en un nodo, y puede migrar de nodo a otro.

Comportamiento

Son las partes dinámicas de los modelos de UML. Éstos son los verbos de un modelo, y representan comportamiento en el tiempo y espacio.

* 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. El comportamiento de una sociedad de objetos o una operación individual puede especificarse con una interacción. Una interacción involucra muchos otros elementos. Mensajes, secuencias de acción (el comportamiento involucrado por un mensaje) y enlaces (conexiones entre objetos).
* Máquina de estados: es un comportamiento que especifica las secuencias de estados por las que pasa un objeto o una interacción durante su vida en respuesta a eventos, junto con sus reacciones a estos eventos. El comportamiento de una clase individual o una colaboración de clases pueden especificarse con una máquina de estados. Una máquina de estados involucra a elementos: estados, transiciones (el flujo de un estado a otro), eventos (que disparan una transición) y actividades (la repuesta a una transición).

Agrupación

Los elementos de agrupación son las partes organizativas de los modelos de UML. Estos son las cajas en las que puede descomponerse un modelo.

* Paquete: es un mecanismo de propósito general para organizar elementos en grupos. Los elementos estructurales, e incluso otros elementos de agrupación pueden incluirse en un paquete.

Anotación

Los elementos de anotación son las partes explicativas de los modelos UML. Son comentarios que se pueden aplicar para describir, clarificar y hacer observaciones sobre cualquier elemento de un modelo.

* Nota: es simplemente un símbolo para mostrar restricciones y comentarios junto a un elemento o una colección de elementos.

Relaciones

Dependencia

Es una relación semántica entre dos elementos, en la cual un cambio a un elemento (el elemento independiente) puede afectar a la semántica del otro elemento (elemento dependiente).

Asociación

Es una relación estructural que describe un conjunto de enlaces, los cuales son conexiones entre objetos. La agregación es un tipo especial de asociación, que representa una relación estructural entre un todo y sus partes.

Generalización

Es una relación de especialización/generalización en la cual los objetos del elemento especializado (el hijo) pueden sustituir a los objetos del elemento general (el padre). De esta forma, el hijo comparte la estructura y el comportamiento del padre.

Realización

Es una relación semántica entre clasificadores, en donde un clasificador especifica un contrato que otro clasificador garantiza que cumplirá. Se pueden encontrar relaciones de realización en dos sitios: entre interfaces y clases y componentes que las realizan, y entre los casos de uso y las colaboraciones que los realizan.

Diagramas

Un diagrama es una representación gráfica de un conjunto de elementos, visualizado la mayoría de las veces como un grafo conexo de nodos (elementos) y arcos (relaciones).

Diagrama de clases

Muestra un conjunto de clases, interfaces y colaboraciones, así como sus relaciones. Estos diagramas son los diagramas más comunes en el modelado de sistemas orientados a objetos. Los diagramas de clases cubren la vista estática de un sistema. Incluyen clases activas que cubren la vista de procesos estática de un sistema.

Diagrama de objetos

Muestran un conjunto de objetos y sus relaciones. Representan instantáneas de instancias de los elementos encontrados en los diagramas de clases. Cubren la vista de diseño estática, pero desde la perspectiva de casos reales.

Diagrama de casos de uso

Muestra un conjunto de casos de uso y actores (un tipo especial de clases) y sus relaciones. Cubren la vista estática de un sistema. Son especialmente importantes en el modelado y organización del comportamiento del sistema.

Diagrama de interacción

Muestra una interacción, que consta de un conjunto de objetos y sus relaciones, incluyendo los mensajes que pueden ser enviados entre ellos. Cubren la vista dinámica de un sistema.

Diagrama de secuencia

Es un diagrama de interacción que resalta la ordenación temporal de los mensajes.

Diagrama de colaboración

Es un diagrama de interacción que resalta la organización estructural de los objetos que envían y reciben mensajes.

Entonces, los diagramas de secuencia y los de colaboración podemos decir que son isomorfos, porque se pueden tomar uno y transformarlo en otro.

Diagrama de transición de estados

Muestra una máquina de estados, que consta de estados, transiciones, eventos y actividades Cubren la vista dinámica de un sistema. Son importantes en el modelado del comportamiento de una interfaz, una clase o una colaboración.

Diagrama de actividades

Es un tipo especial de diagrama de estados que muestra el flujo de actividades dentro de un sistema. Cubren la vista dinámica de un sistema. Sirven para modelar el funcionamiento de un sistema y resaltan el flujo de control entre objetos.

Diagrama de componentes

Muestra la organización y dependencias entre un conjunto de componentes. Cubren la vista de implementación estática de un sistema. Se relacionan con los diagramas de clases en que un componente se corresponde, por lo general, con una o más clases, interfaces o colaboraciones.

Diagrama de despliegue

Muestra la configuración de nodos de procesamiento en tiempo de ejecución y los componentes que residen en ellos. Cubren la vista de despliegue estática de una arquitectura. Se relacionan con los diagramas de componentes en que un nodo incluye, por lo general, uno o más componentes.

Descripción de la herramienta: Rational

Rational es una de las más poderosas herramientas de modelado visual para el análisis y diseño de sistemas basados en objetos. Soporta especificación de requisitos mediante casos de uso, análisis y diseño orientado a objetos utilizando UML, round trip engineering en Java, C++ y Visual Basic, etc. y se integra con productos como ERWin y Oracle 8. Esta herramienta soporta el enfoque y perspectiva de UML.

Se utiliza para modelar un sistema antes de proceder a construirlo. Usando una herramienta de este tipo, el equipo de desarrollo puede comunicar efectivamente la arquitectura del software por representación gráfica, documentar y generar el código de la aplicación.

Rational cubre todo el ciclo de vida de un proyecto: Concepción y formalización del modelo, construcción de los componentes, transición a los usuarios y certificación de las mismas fases.

Con Rational se pueden utilizar los diagramas para visualizar el sistema desde diferentes perspectivas. Estos diagramas pueden tener cualquier combinación de elementos y relaciones. Hay que tener en cuenta que en la práctica solo surge un pequeño número de combinaciones, las cuales son consistentes con las vistas más útiles que comprenden la arquitectura de un sistema con gran cantidad de software. Rational incluye, entre otros, los siguientes diagramas: de clases, de objetos, de Casos de Usos, de Secuencia, de Colaboración, de Estados, de Actividades, de Componentes, de Despliegue.

Además de prestar soporte a la notación de la metodología UML, la herramienta ofrece también modelar con la notación de OMT y BOOCH.

Ahora bien, que beneficios trae aparejado de esta herramienta de modelado visual son las siguientes:

• Ciclos de desarrollos cortos por medio de un desarrollo interactivo controlado.

• Incremento de la productividad por medio del desarrollo del "proceso dirigido por casos de uso" y eliminación del riesgos.

• Soporte para proyectos de gran tamaño y organizaciones con múltiples proyectos por medio de un ambiente escalable.

• Significante reutilización del software a través de software basado en arquitectura y componentes.

• Mejora la comunicación del equipo de trabajo por medio del lenguaje de para modelar.

• Integración con sistemas heredados por medio de las capacidades de ingeniería inversa y extensibilidad de interfaces.

• Soporte de generación de código a lenguajes como: Java, C++, Visual Basic, Power Builder, Smalltalk.

• Soporte para definición de interfaces de Base de Datos.

La Interfaz de Rational Rose está formada por los siguientes elementos principales:

*Browser o Navegador, que permite navegar rápidamente a través de las distintas vistas del modelo.

*Ventana de Documentación, para manejar los documentos del ítem seleccionado en cualquiera de los diagramas.

*Barra de Herramientas Standard, para acceder rápidamente a las acciones comunes a ejecutar para cada uno de los diagramas del modelo.
*Barra de Herramientas Diagrama, muestra el conjunto de herramientas disponibles para el diagrama activo.
*Ventana de Diagrama, que permite desplegar y editar cualquiera de los diagramas UML.
*Ventana Registro o Log, que registra todas las órdenes ejecutadas y los errores que se producen durante su ejecución.
*Barra de Estado, que muestra el programa de la carga del modelo, el estado de lectura/escritura del elemento seleccionado, y otros datos de utilidad.
En la figura 9 se muestra la interfaz de Rational de un proyecto:

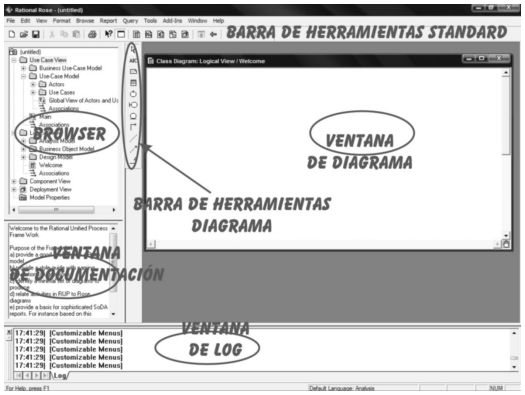


Figura 9: Interfaz de un proyecto en Rational

En la figura 10 se muestra el browser de un proyecto:

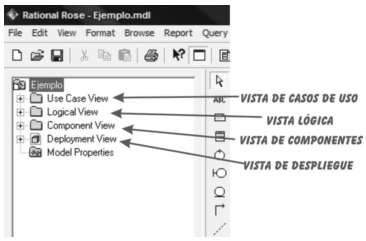
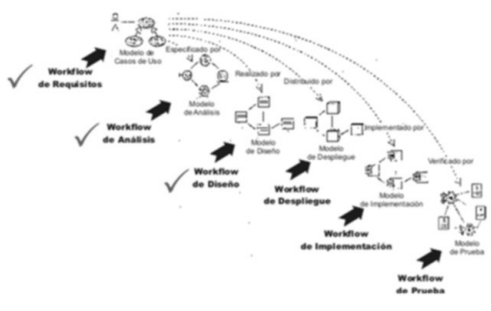


Figura 10: Browser de un proyecto en Rational

¿Cómo se procedió para modelar las Gramáticas Formales y Máquinas Abstractas?

La vida del proceso unificado se repite a lo largo de una serie de ciclos que constituyen la vida de un sistema. Cada ciclo constituye una versión del producto.

Figura 11: Ciclo de vida de un producto



Para llevar a cabo este ciclo de vida de manera eficiente, los desarrolladores necesitan todas las representaciones del producto de software:
En la presente proyecto de investigación, correspondiente al modelado de Gramáticas Formales y Máquinas Abstractas con la metodología RUP/UML con la utilización de la herramienta Rational, se hizo más hincapié en tres workflows (tildados), ya que son los que permiten realizar una especificación de requerimientos del objeto bajo estudio, y se detallarán a continuación.

Workflow de Requisitos

Los propósitos de este workflow son:

- Dirigir el sistema para lograr un sistema correcto.
- Ingeniería de requerimientos.
- Permite descubrir, analizar, validar y documentar los servicios y restricciones del sistema.

Puntos de Partida en este proyecto

- Modelo del Dominio del Problema (MODP) de Gramáticas Formales y Máquinas Abstractas, cuyas características son:

  Parte del usuario

  El que decide que va a hacer es el jefe del proyecto junto al analista.

  En el análisis permite detectar mas responsabilidades; y en el diseño permite ver cómo se pasan los mensajes los objetos entre sí (diagrama de secuencia).

  Modelo del Dominio del Problema (MODP) de Gramáticas Formales y Máquinas Abstractas, cuyas características son:

  Especificación de requerimientos.

- Noción sobre gramáticas formales y máquinas abstractas (Apuntes teóricos de la Cátedra de Sintaxis y Semántica del Lenguaje).

Basándose en los ejercicios que figuran en las guías de trabajos prácticos se fueron identificando los casos de uso esenciales del sistema, y algunos casos de uso de soporte;

luego fueron descriptos a trazo fino, siguiendo la modalidad de cómo se resuelven los distintos ejercicios. En la descripción, además de tener en cuenta el procedimiento de resolución se tuvo en cuenta las clases, atributos y métodos del diagrama de clases realizado en primera instancia (MODP).

Workflow de Análisis

Los propósitos de este workflow son:

-Analizar, refinar y estructurar requisitos identificados en el workflow de requisitos.
-Entrada: requerimientos funcionales y algunas especificaciones especiales o adicionales.

¿Por qué es importante el análisis?

Ayuda a comprender los requisitos funcionales.

Puede facilitar la capacitación de la gente que tiene que trabajar en el proyecto.

Sirve como modelo conceptual sobre gramáticas formales y máquinas abstractas para la implementación.

En este proyecto, se trabajó con clases de análisis, que representan la abstracción de una/varias clases y/o subsistemas del diseño del sistema. Se centran en los requerimientos funcionales, posponiendo los requerimientos no funcionales (llamándolos requisitos especiales). Define más atributos, participa en relaciones. Trabaja en tres dimensiones: comportamiento, información y presentación.

Las clases de análisis fueron modeladas porque obtienen elementos de la solución, agregan atributos, asignan responsabilidades. Para encontrar nuevas responsabilidades se utilizó el diagrama de colaboración.

El diagrama de colaboración muestra cómo están organizados (estructura) los objetos, y cómo es el paso de los mensajes entre ellos. Modela: vista interna del sistema, clases, comportamiento, realizaciones de casos de uso y colaboraciones entre objetos. Permite refinar el diagrama de clases, ya que permite identificar nuevos atributos y métodos. Para construirlo se utiliza el diagrama de clases y el diagrama de casos de uso. No tiene en cuenta el paso de mensajes a través del tiempo (diferencia con el diagrama de secuencia).

Workflow de Diseño

Los propósitos de este workflow son:
-Refinar requerimientos no funcionales y trabajar con las restricciones tecnológicas (ambiente de implementación).
-Identifica subsistemas.
-Identifica las interfaces que van a existir entre los subsistemas.
-Esboza la implementación del sistema.
-Genera artefactos o elementos que facilitan la implementación.

Las salidas de este workflow es el Modelo de Diseño, que se basa en “Niveles de Diseño”:

*Nivel sistema: hardware+software+comunicación=diagrama de despliegue
*Nivel de Casos de Uso: Se efectúan realizaciones de casos de uso (se utiliza: diagrama de secuencia, diagrama de componentes, diagrama de transición de estados). En este proyecto se utilizó el diagrama de secuencia.
*Nivel de clase: Se trabaja a nivel de atributos (tipos de datos, longitud, etc.) depende de la decisión del arquitecto del lenguaje de programación. Utiliza el diagrama de transición de estados y el diagrama de actividades.

En cuanto a los diagramas de secuencia, en el fueron construidos a partir de los diagramas de colaboración (gracias a la herramienta utilizada), pero podría haber sido generado también a partir del diagrama de clases de diseño y la descripción del caso de uso analizado. Un diagrama de secuencia hace énfasis en el orden cronológico del paso de mensajes (métodos que formarán parte del diagrama de clases de diseño). Modela comportamiento. Modela la vista interna del sistema. Es un diagrama dinámico. Trabaja con objetos. Permite modelar colaboraciones entre objetos.

Resultados

Como resultado del trabajo de investigación realizado, se elaboró un esquema conceptual sobre un dominio de aplicación determinado: Máquinas Abstractas y Gramáticas Formales

que fue modelado con la metodología Rational Unified Process con la utilización de la herramienta Rational. El mismo tiene doble finalidad: por un lado servir como caso de estudio en las asignaturas donde se desarrollan contenidos metodológicos en el diseño de sistemas de información, más concretamente sobre contenidos de especificación de requerimientos; pero otro lado, proveer una conceptualización sobre el dominio de aplicación que servirá para impartir conocimientos sobre los contenidos de informática Teórica: transferir las técnicas resultantes de mejores prácticas en lo que se refiere a la especificación de requerimientos a empresas de desarrollo de software de la región y al perfeccionamiento de los docentes.

Discusión

La aplicación de la metodología RUP/UML y la utilización de la herramienta Rational para el modelado de un esquema conceptual resulta un medio adecuado para la explicación clara de y sencilla de las Gramáticas Formales y Máquinas Abstractas. Al estar orientado a un perfil educativo, resulta especialmente fácil visualizar cómo se modelaron las Gramáticas Formales y Máquinas Abstractas, tema que es bastante abstracto y costoso de comprender. El trabajo posee los diagramas esenciales de la aplicación de la metodología RUP utilizando UML para las notaciones gráficas y para el modelado la conocidísima herramienta Rational, puesto que detrás de cada diagrama se destaca la funcionalidad principal de un sistema de información que fue detectada en base a los casos de uso identificados durante el desarrollo de este proyecto de investigación. De este modo lo que se ha conseguido la comprensión del tema de modo gráfico, intuitivo y sencillo mediante visualización del modelo desarrollado con RUP.

Conclusión

El primer tema desarrollado en este trabajo fue la descripción exhaustiva de las Gramáticas Formales y las Máquinas Abstractas que serán la entrada del modelo desarrollado con RUP/UML, con la ayuda de la poderosa herramienta Rational.

Por otro lado, se procedió a realizar un profundo estudio sobre la metodología Rational Unified Process y la herramienta Rational a utilizar para el modelado. Teniendo bien en claro los objetivos y todos los conceptos a manejar, se procedió a modelar las Gramáticas Formales y Máquinas Abstractas; cuyo resultado es un modelo que permitirá un mejoramiento en la especificación de requerimientos.

Por ultimo, la ventaja de que las Máquinas Abstractas y las Gramáticas Formales sean modelos isomorfos entre sí, permite un realizar conversores para que se pueda pasar de un autómata a una gramática de acuerdo a su tipo; y también realizar el proceso inverso; teniendo entre conversión y conversión modelos equivalentes.

Agradecimientos

El presente trabajo de investigación ha sido realizado en el marco del proyecto de Investigación bajo la dirección del Ing. Marcelo Marciszack.

Referencias

[1] Jacobson, I., Booch, G., Rumbaugh J., El Proceso Unificado de Desarrollo de Software, 2000 Addison Wesley.
[2] Kruchten, P., The Rational Unified Process: An Introduction, 2000 Addison Wesley.
[3] Kruchten, P. Architectural Blueprints—The “4+1” View Model of Software Architecture. IEEE Software 12 (6), November 1995, pp. 42-50.
[4] Rational Software Corporation, Product: Rational Software Corporation, 2002.
[5] Rational Software Corporation, Rational Unified Process. Best Practices for Software Development Teams, 1998.
[6] Páginas de Internet.

Datos de Contacto:

Melisa Daniela Torres Díaz. Universidad Tecnológica Nacional, Facultad Regional Córdoba. La Rioja 5685, Barrio Quebrada de las Rosas, Córdoba Capital. C.P.: 5003.
TE: (0351) 156012379
E-mail: td.melisa@gmail.com