H-Case

Una herramienta CASE basada en un repositorio con interfaz hipermedial.

Directores: Lic. Analía Amandi y Dr. Gustavo Rossi

Autores: Dario Fabini y Gustavo Sobota

U.N.L.P - La Plata, Argentina

Indice

Introducción

Motivación

Los ambientes CASE (Computer Aided Software Egineering) soportan las diferentes actividades involucradas en el desarrollo de software. El uso de sistemas CASE en el desarrollo de sistemas ha crecido considerablemente y nuevos requerimientos han aparecidos para estos sistemas, tal como la habilidad de explorar y integrar diferentes medios. Los sistemas de hipertexto ofrecen una alternativa porque tratan de imitar el asociativo pensamiento humano y ofrece una solución posible para manejar varios tipos de medios[Oinas '92].

Los sistemas de hipertexto de software [Garg '89] son aquellos hipertextos en los cuales sus nodos contienen y organizan información pertinente al desarrollo, uso y mantenimiento de sistemas de software. Así, un sistema que combine la potencia y claridad conceptual de los hipertextos con otras herramientas de apoyo puede proveer un ambiente integrado de ingeniería de software.

De lo anterior surge la idea de combinar facilidades típicas de los ambientes CASE, tales como control de consistencia y completitud de documentos formalizados, provisión de moldes para algunos documentos, manejo de versiones y fundamentalmente modelización de todo el proceso de desarrollo, con el concepto de hipertexto, agregando una facilidad adicional de almacenamiento y recuperación habitualmente ausente en los sistemas de ingeniería de software.

Los conceptos de orientación a objetos proveen las propiedades mas importantes encontradas en varios modelos de sistemas, las cuales permiten a los modelos ser comprehensivos, cambiables, adaptables y reusables [JAC '95].

La utilización de Frameworks en la construcción de software potencia las características de reusabilidad y adaptabilidad de los objetos.

Las herramientas de sofware hacen gran uso de "Metadatos" descripciones de procesos de negocio, interface de las aplicaciones, esquemas de bases de datos, diagramas de ingeniería, configuraciones de software y librerías de documentos. Las herramientas necesitan almacenar estos "Metadatos". Un Repositorio es un sistema de base de datos diseñando para manejas estos "Metadatos". La habilidad de compartir datos entre distintas herramientas depende de que las herramientas puedan usar un repositorio común.[Ber '96].

La combinación de herramientas hipermediales, objetos, frameworks y repositorios en la construcción de herramientas CASE debe potenciarla con las características mencionadas para cada una de estas tecnologías produciendo una herramienta de ingeniería de software muy potente.

Objetivos

Inicialmente el objeto del proyecto es estudiar los ambientes de Ingeniería de Software Orientado a Objetos sobre un Framework de Hipermedias. Como objetivo principal se propuso la construcción de un ambiente CASE del tipo mencionado, y se marcaron ciertas pautas para su construcción.

Construirlo con tecnología de objetos

Teniendo en cuenta la gran cantidad de requerimientos que han aparecido para estos ambientes se espera obtener una herramienta lo suficientemente adaptable y extensible para evolucionar de un prototipo a un ambiente que cubra un gran número de requerimientos

Construirlo sobre un framework de hipermedias

En este sentido se pretende demostrar el potencial de un framework de hipermedias y capitalizar el esfuerzo de construcción del framework para obtener un prototipo potente con un menor esfuerzo.

Se pretende trasladar la capacidad de adaptabilidad y extensibilidad del framework a la herramienta.

Además se desea mostrar que la potencia y claridad conceptual de la interfaz hipermedial puede ayudar a satisfacer muchos de los requerimientos de las herramientas CASE. Se puede obtener riqueza expresiva, formas potentes de estructurar información y otras ventajas que serán analizadas oportunamente.

Poder ejcutarlo

El tercer objetivo es obtener un prototipo ejecutable y no solamente un diseño, esta es la única forma de mostrar la viabilidad de una herramienta de este tipo.

Trabajo realizado

A partir de las primeras definiciones se plantearon una serie de objetivos intermedios.

Dado que al comienzo del proyecto contábamos solo con conocimientos de programación orientada a objetos y Smalltalk. En su mayoría, los objetivos intermedios tienen que ver con la adquisición del conocimiento necesario de conceptos y las herramientas, para alcanzar los objetivos iniciales del proyecto.

Los objetivos intermedios mas importantes fueron:

- Estudio del dominio y requerimientos de las herramientas de ingeniería de software.
- Estudio de Metodologias OO para basar el prototipo(Wirf, Jacobson, OBA, etc.).
- Selección y aprendizaje de las herramientas para la construcción del prototipo

  Object Works

  Metodologias OO para desarrollar el prototipo

  Framework de hipermedia

  HotDraw

  Repositorios

  MVC
- Diseño, construcción y prueba del prototipo
- Elaboración del informe
- Evaluación general y conclusiones

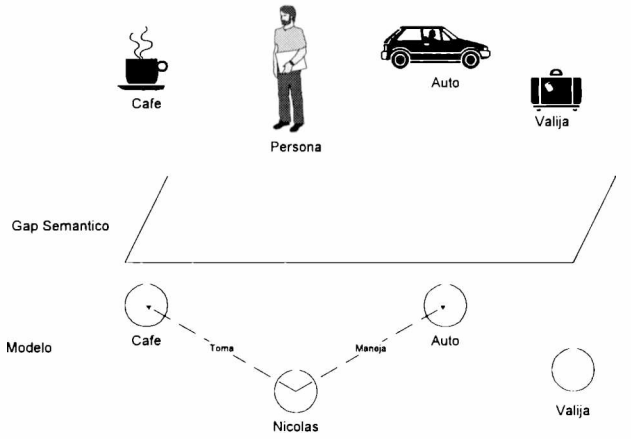
Estos objetivos intermedios fueron organizados en etapas que, a medida que se cumplieron, se transformaron en las distintas tares realizadas a lo largo del trabajo.

Base Conceptual

Objetos

Orientación a objetos

La orientación a objetos es un aproximación especial para la construcción de modelos de sistemas complejos, en la cual un sistema complejo, que consiste de un gran numero de ocurrencias, es visto como un conjunto de objetos. Las relaciones entre dichas ocurrencias son vistas como asociaciones entre objetos; sus propiedades como atributos de los objetos. Además, las ocurrencias pueden tener características dinámicas o estáticas. Una ocurrencia que afecta a otra cuando cierto evento tiene lugar es descripta como una comunicación entre objetos.



Cuando queremos modelar un problema real, en general, nos encontramos con eventos y ocurrencias reales, que pueden ser materiales o inmateriales pero que son lo suficientemente tangibles como para tener un precio, poder ser producidas o poder ser medidas de alguna forma. Todas estas ocurrencias son modeladas como *objetos*. Como hay una relación cercana entre la ocurrencia de la vida real y el objeto en el modelo el gap semántico se reduce.

No solamente las ocurrencias se describen como objetos, las relaciones entre objetos son mostradas como *asociaciones*. Esto permite representar la estructura estática de las ocurrencias y sus relaciones en un modelo, pero la orientación a objetos también permite a las ocurrencias que cambian dinámicamente ser modeladas como objetos. Por lo tanto los objetos pueden demostrar comportamiento respecto de su ambiente, pueden ser activados, pueden cambiar y pueden activar a otros objetos.

Objetos

Los objetos encapsulan información y pueden ofrecer comportamiento a otros objetos. La información es capturada por lo que llamamos *atributos,* el comportamiento es capturado por las *operaciones* que puede ofrecer a otros objetos en su ambiente. Un objeto puede ser alcanzado por otros objetos mediante sus operaciones. Un objeto usa a otro enviándole un mensaje, dependiendo del mensaje que recibe el objeto receptor selecciona alguna de sus operaciones y la ejecuta. Mientras ejecuta la operación un objeto puede cambiar los valores de sus atributos o decidir hacer una cosa u otra, incluso puede mandar un mensaje a otro objeto o a si mismo.

La única forma de alcanzar la información en un objeto es usando sus operaciones, esta propiedad es llamada *encapsulamiento*. No solo la información esta oculta, el detalle de como se realizan las operaciones también esta oculto. Los nombres y los tipos de información que cada operación requiere son llamados la *signatura* de la operación. La combinación de signaturas que un objeto puede ofrecer es llamada el *protocolo* del objeto.

Podemos definir la colaboración entre objetos en términos de *contratos*. Un contrato especifica los mensajes que pueden ser enviados entre dos objetos. Un objeto que los envía (Client) y otro que los recibe (Server). Los requerimientos ubicados en el server son llamados *responsabilidades*.

Por lo antes dicho es posible describir a un objeto en dos formas diferentes aunque relacionadas. El protocolo o especificación del objeto y la descripción interna (atributos y detalle de operaciones) o implementación del objeto. La ventaja de esto es que el ambiente del objeto no es afectado por un cambio en la implementación de un objeto mientras no se afecte el protocolo.

Clase

Un objeto pertenece a una *clase* de objetos similares. Una clase es un Template para los objetos y define para que pueden ser usados. La clase describe las operaciones que el objeto puede ofrecer, las varias actividades que pertenecen a dicha operación y los distintos atributos que los objetos deben tener. Cada objeto tiene información de la clase a la que pertenece y realiza sus operaciones de acuerdo a lo definido para la operación por su clase.

Un objeto asume diferentes valores para sus atributos, durante su tiempo de vida los cambios en los atributos de un objeto ocurren a medida que se van ejecutando operaciones sobre él. Las mismas operaciones son comunes a todos los objetos pertenecientes a la misma clase. Cuando un objeto es creado conforme a la descripción de una clase se dice que es una *instancia*. Una clase que tiene al menos una instancia se dice *instanciada*.

Herencia

Una clase puede heredar de otra. Esto quiere decir que una instancia de la clase heredera recibirá todas las operaciones y atributos de la clase heredera y de la clase heredada. El uso de la herencia es otro ejemplo de reuso. Si dos o mas clases tienen propiedades similares, las propiedades en común pueden ser factorizadas y descriptas en una clase llamada abstracta (Porque nunca será instanciada). Este acto es llamado *generalización*.

Otro tipo de uso de la herencia envuelve el reuso de una clase mas general, agregando o cambiando una o mas de sus propiedades. Este tipo de herencia es comúnmente llamado *especialización*. Cuando se crea una nueva clase es posible querer reusar propiedades de mas de una clase. Una forma de manejar esto es permitir heredar propiedades de mas de una clase esto es llamado *herencia múltiple*. Una desventaja de la herencia múltiple es la tendencia a hacer que el modelo sea mas dificultoso de comprender dado que pueden aparecer conflictos.

El mecanismo de herencia hace que el modelo de un sistema pueda absorber cambios con mayor facilidad.

Polimorfismo y Binding dinámico

Cuando hablamos de programación orientada a objetos surgen dos conceptos fundamentales: Polimorfismo y "binding".

Polimorfismo significa que el emisor del mensaje no necesita conocer la clase de la instancia receptora. El transmisor provee solo un pedido de la operación especificada, mientras que el receptor conoce cómo realizar esta operación.

La conexión entre el mensaje recibido y la operación apropiada se conoce como "Binding". Si el binding se produce durante la compilación se llama binding estático. La característica polimórfica del lenguaje hace que a veces sea imposible conocer en tiempo de compilación la clase del receptor y de esta manera decidir que operación ejecutar. Si el binding ocurre cuando el mensaje es enviado realmente, es decir en tiempo de ejecución, se dice que se hace un binding dinámico.

La ventaja con el binding dinámico es la flexibilidad del sistema obtenido. Por otro lado el binding estático es mas seguro y eficiente. Es mas seguro si tenemos un lenguaje tipado, porque algunos errores son notados en tiempo de compilación y no causan fallos en tiempo de ejecución.

El binding dinámico es una forma de implementar polimorfismo. Pero el polimorfismo puede ser implementado aun sin binding dinámico. Este es el caso en el que tenemos un puntero a una instancia, el tipo de la variable es padre de la clase de la instancia, la operación a ser realizada esta declarada en esta clase padre y no fue sobrescrita en ningún descendente.

Beneficios

Cuando utilizamos estos conceptos para construir sistemas, obtenemos productos comprensibles, entendibles, cambiables, adaptables y reusables.

- *Comprehensible*: Podemos mirar las clases jerárquicamente y obtener un panorama de todo el negocio que estamos modelando.
- *Entendible:* El negocio es descripto en término de objetos que, a menudo, son encontrados en la realidad.
- *Cambiable*: Los cambios son usualmente locales a una clase dada. Un buen diseño permite que cambios mas importantes afecten pocas clases.
- *Adaptable*: El mecanismo de especialización permite adaptar los sistemas a diferentes situaciones.
- *Reusable*: Las clases pueden ser construidas y manejadas como componentes, además cuando creamos nuevas clases podemos reusar propiedades de clases que ya han sido definidas.

Metodologías

Existen varios métodos de desarrollo orientado a objetos. Y el número sigue creciendo como resultado del gran interés que existe en las técnicas orientadas a objetos. Algunos de ellos son:

Object-Oriented Design (OOD) de Grady Booch (1991).

Object-Oriented System Analysis (OOSA) de Shaler y Mellor (1988)

Object Modeling Technique (OMT) de Rumbaugh (1991)

Object-Oriented Analysis (OOA) de Peter Coad y De Yourdon (1991)

Hierarchical Object-Oriented Design (HOOD) (1990) inicialmente desarrollada por la European Space Agency (ESA) y luego desarrollada por el HOOD Working Group.

Object-Oriented Structured Design (OOSD) de Wasserman (1990).

Responsibility-Driven Design de Wirf-Brock (1990).

Object-Oriented Role Analysis, Syntesis and Structuring (OORASS).

Object-Oriented Systems Analysis (OSA) de Embley (1992).

Object Behavior Analysis (OBA) de Goldberg y Rubin.(1992)

Object-Oriented Software Engineering de Jacobson, Christerson, Jonsson y Overgaard. (1992)

Las metodologías tienen varios conceptos en común, aunque a veces los nombran diferentes, algunos de ellos son:

- Clase.
- Objeto.
- Herencia.
- Comunicación, colaboración.
- Estímulo, mensaje, evento.
- Método, servicio, operación.
- Atributo.
- Escenario, caso de uso.
- Subsistema.
- Contrato, interfaz provista.
- Usuario, actor, objeto del ambiente.
- Block, Módulo.

Metodología de Jacobson

Introducción

Object-Oriented Software Engineering de Jacobson, Christerson, Jonsson y Overgaard enfatiza el concepto de "Desarrollo de sistema" como forma de brindar soporte de computación para una organización o partes de ella, y que bajo esta premisa es de suma importancia ver el sistema desde la perspectiva de la organización y del usuario.

La base de la metodología se encuentra en tres técnicas:

1. Object-oriented programming
2. Conceptual modelling (Hull and King (1987), Tsichritzis and Lochovsky(1982) or Brodie et al. (1984))
3. Block design

La metodología prevé la existencia de una especificación de requerimientos escrita de alguna forma. El alcance incluye todo el ciclo de vida del sistema, hasta que es reemplazado por otro. Esto incluye el desarrollo inicial, desarrollos posteriores y todo el mantenimiento del sistema.

Para manejar la complejidad del sistema organizadamente, propone la construcción de varios modelos, cada uno enfocando cierto aspecto del sistema. De esta forma pretende introducir la complejidad gradualmente y en un orden específico.

Los modelos propuestos son:

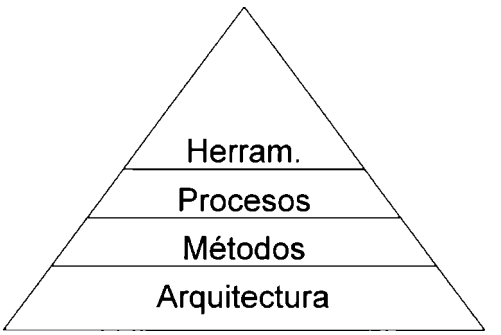
- *Requirements model* cuyo objetivo es capturar los requerimientos funcionales.
- *Analysis model* cuyo objetivo es darle al sistema una estructura de objetos robusta y modificable.
- *Design model* cuyo objetivo es adaptar la estructura del modelo de análisis al ambiente de implementación.
- *Implementation model* cuyo objetivo es implementar el sistema.
- *Test model* cuyo objetivo es verificar el sistema

además sugiere la inclusión de algún tipo de modelo útil de acuerdo a la aplicación.

Estos modelos son salida de las actividades de:

- Analysis
- Construction
- Testing
- Components

Arquitectura



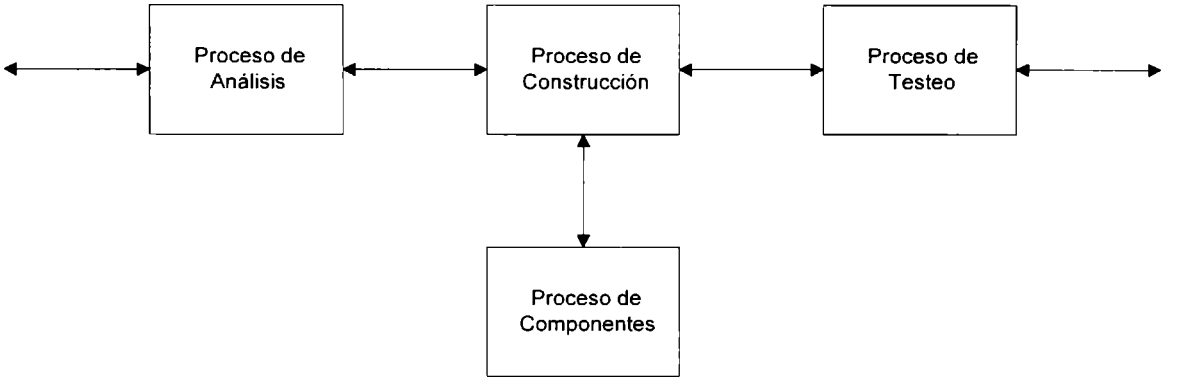
Para su construcción, cada modelo necesita un lenguaje de modelación, notación o técnica de modelización. Jacobson dice que este conjunto de técnicas de modelización define la *arquitectura* sobre la que el desarrollo del sistema está basado. Más formalmente, define que la arquitectura de un método de desarrollo de sistemas es la detonación de su conjunto de *técnicas de modelización.*

Una técnica de modelización generalmente se describe a través de sus *sintaxis* (Como se ve), *semántica* (Que quiere decir) y *pragmática* (Heurísticas y recomendaciones para el uso de la técnica).

Para construir esos modelos además se necesitan *métodos*, que muestren cómo trabajar con las técnicas de modelización. Los métodos permiten la utilización de procedimientos de manera de realizar paso a paso y en forma planeada la construcción de los modelos.

Por otro lado se visualiza el desarrollo como un producto y se divide el trabajo de desarrollo en procesos que manejan el producto. Cada proceso describe una actividad sobre el producto e interactúa intensivamente con los otros procesos. A su vez, cada proceso puede ser descompuesto en otro conjunto de procesos comunicantes. Los procesos son la versión a gran escala de los métodos, un método tiene utilidad en un desarrollo, los procesos tienen utilidad en el concepto de producir gran cantidad de desarrollos, éste es el caso de una empresa con proyectos para hacer nuevos sistemas y proyectos para mantener sistemas existentes.

Cada proceso puede ser soportado por herramientas. Estas herramientas son esenciales, especialmente cuando el desarrollo es a gran escala. Con la ayuda de herramientas se puede automatizar parte del trabajo y obtener una ayuda invaluable en el manejo de la consistencia.



El *proceso de desarrollo de sistemas* se compone de varios procesos:

En el *proceso de análisis* se crea una descripción conceptual del sistema, su objetivo es comprender el sistema, comunicarse con quien ordenó el sistema y con el proceso de construcción.

En el *proceso de construcción* se desarrolla el sistema desde los modelos creados dentro del proceso de análisis. Este proceso finaliza con el sistema construido en su totalidad.

El *proceso de testeo* integra el sistema, lo verifica y decide cuándo puede ser puesto en producción.

Aparte de estos 3 procesos principales, existe el *proceso de generación de componentes*, el cual se comunica principalmente con el proceso de construcción, este proceso desarrolla y mantiene componentes que luego son usadas en el proceso de construcción.

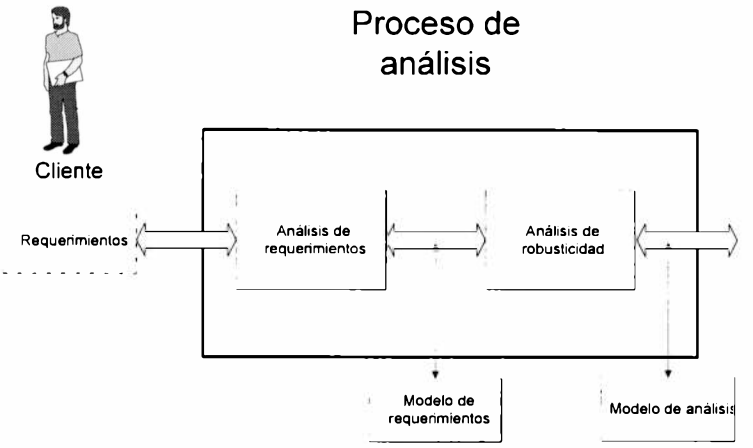
Los procesos y los modelos están relacionados íntimamente. Cada proceso toma uno o más modelos y los transforma en otros modelos. El producto final debe ser una descripción completa y testeada del sistema. Esta descripción normalmente se compone de documentación y código fuente. Podemos mirar la especificación de requerimientos como un modelo. Con esta hipótesis podemos decir que la entrada y salida de los procesos son modelos.

El proceso de análisis produce dos modelos. A partir de la especificación de requerimientos, se crea un modelo de requerimientos. El modelo de requerimientos sirve como base para otro modelo creado en el proceso de análisis, el modelo de análisis.

El proceso de construcción se basa en los modelos anteriores y elabora, primero el modelo de diseño y luego el modelo de implementación. El modelo de implementación, junto con el de requerimientos, proveen la entrada para el proceso de test. Este proceso produce el modelo de test, que es el resultado de testear el modelo de implementación contra los requerimientos. El trabajo de desarrollo fluye sobre estos procesos cuando interactúan uno con el otro.

Arquitectura de modelos

Modelo de requerimientos



El modelo de requerimientos consiste de:

- Un modelo de casos de uso
- Descripciones de interfaces
- Un modelo de dominio del problema

El *modelo de casos de uso* usa Actores y Casos de uso. Estos conceptos definen lo que está fuera del sistema (Actor) y lo que debe ser realizado por el sistema (Caso de uso). Los *Actores* modelan cualquier cosa que necesita cambiar información con el sistema. Los actores pueden modelar usuarios humanos, pero también modelan otros sistemas que se comunican con el sistema.

Jacobson diferencia entre Actor y Usuario, un Actor representa cualquier cosa que necesita intercambiar información con el sistema no necesariamente un usuario, además un actor representa un rol y un usuario en particular puede jugar distintos roles, es decir puede ser representado por más de un actor. Las acciones del actor son no determinísticas.

Los actores que usan el sistema directamente son llamados *actores primarios*. Estos actores realizan las tareas principales del sistema. Además de los actores primarios hay actores que supervisan y mantienen el sistema, estos son llamados *actores secundarios*.

Una instancia de un actor realiza un número de operaciones diferentes sobre el sistema. Cuando utiliza el sistema realiza una secuencia de transacciones sobre el sistema, relacionadas por el comportamiento, esa secuencia de transacciones se llama *caso de uso*.

Un Caso de uso es una forma específica de usar el sistema utilizando alguna parte de su funcionalidad. Cada caso de uso constituye un curso completo de eventos iniciado por un actor y especifica la interacción que tiene lugar entre el actor y el sistema. Un caso de uso es una secuencia especial de transacciones relacionadas

realizadas por un actor y el sistema en un diálogo. La colección de todos los casos de uso especifican todas las formas existentes de utilizar el sistema.

Una transacción termina cuando el sistema se queda nuevamente esperando un estímulo del actor.

Para construir un caso de uso primero se describe el curso básico. El curso básico es el curso de eventos más importante que da mejor comprensión del caso de uso. Las variantes del curso básico de eventos, y errores que puedan ocurrir, se describen en cursos alternativos.

Un concepto utilizado para estructurar y relacionar descripciones de casos de uso es la asociación *extensión*. La extensión especifica cómo una descripción de caso de uso puede ser insertada en otra descripción de caso de uso extendiéndola. El caso de uso cuya descripción insertada debe ser un curso completo en sí mismo.

La extensión se utiliza para modelar casos como:

- Partes opcionales de un caso de uso
- Cursos alternativos complejos que rara vez ocurren.
- Sub-cursos separados que son ejecutados sólo en ciertos casos.
- Situaciones en que varios casos de uso son insertados en un caso de uso especial.

Un posible refinamiento del modelo de casos de uso está dado por la identificación y extracción de partes similares de los casos de uso. Así se debe describir la parte similar una vez, en lugar de repetirla para cada caso de uso que muestre ese comportamiento. Cualquier cambio que experimente dicha parte afectará todos los casos de uso que la comparten. El caso de uso extraído se denomina *caso de uso abstracto* porque nunca será instanciado, mientras que el caso de uso que realmente será instanciado se llama *caso de uso concreto.*

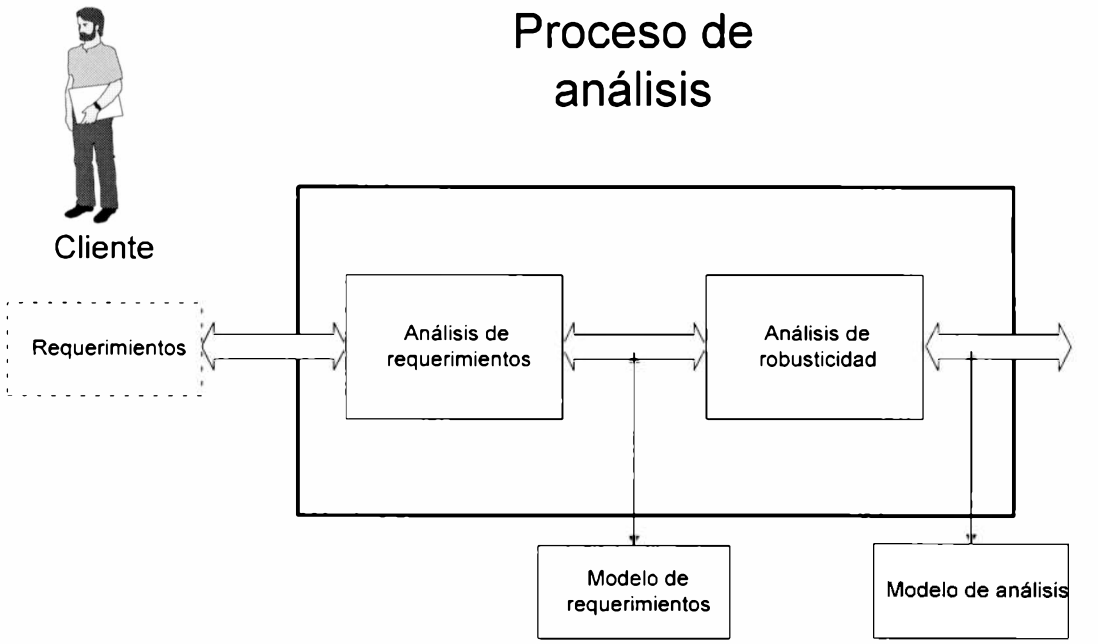
La relación entre el caso de uso abstracto y el concreto se denomina *Usa*. Una técnica para extraer casos de uso abstractos es identificar actores abstractos. Un *actor abstracto* describe un rol que debería ser jugado frente al sistema. Cuando diferentes actores juegan roles similares deberían heredar de un actor abstracto común. Una extensión a un caso de uso puede ser visto como un caso particular de caso de uso que tiene significado propio.

El modelo de casos de uso hace la *delimitación del sistema* definiendo el límite entre actores y casos de uso.

Las *Descripciones de interfaz* son apropiadas como soporte del modelo de casos de uso. Aquí puede ser muy útil una herramienta de prototipación de interfaces.

El *modelo de dominio del problema* también es apropiado como herramienta de comunicación con el usuario potencial. En general es un modelo de objetos del dominio del tipo de la metodología de Wirf.

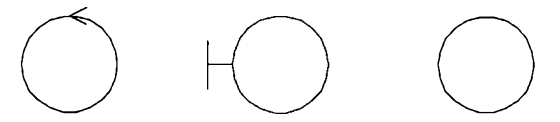
Modelo de análisis



Cuando el modelo de requerimientos ha sido desarrollado y aprobado por quien ordenó el sistema o el potencial usuario, se comienza a desarrollar el sistema. El desarrollo comienza con el modelo de análisis. El modelo de análisis intenta estructurar el sistema en forma independiente del ambiente de implementación, para no agregar la complejidad propia del ambiente de implementación en una etapa tan temprana, y para generar una estructura estable, robusta, mantenible y extensible.

Muchas metodologías utilizan el modelo de objetos del dominio para este fin, pero Jacobson sostiene que el modelo de análisis tiene características que lo hacen mejor para los efectos nombrados. Además sostiene que muchos cambios en el sistema serán debido a cambios en el ambiente de implementación, estos cambios no afectan al modelo de análisis, lo que lo hace más estable. El modelo de análisis se construye especificando objetos. A diferencia de otras metodologías, que sólo utilizan uno, en el modelo de análisis se utilizan tres tipos de objetos:

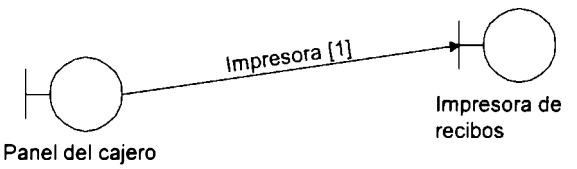
- Objetos entidad
- Objetos interfaz
- Objetos control



Cada tipo de objeto tiene un propósito.

El *objeto interfaz* modela el comportamiento y la información dependiente de la interfaz del sistema. La tarea de un objeto interfaz es transformar la entrada de un Actor en eventos dentro del sistema, y transformar aquellos eventos en el sistema en los que el Actor esté interesado, en algo presentable al Actor y presentarlo. En definitiva pueden describir la comunicación bidireccional entre el Actor y el sistema.

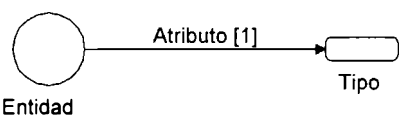
Es muy frecuente que los objetos interfaz dependan unos de otros para poder resolver una tarea. Para modelar esto se introduce una asociación que se llama *acquaintance association* entre objetos. Una asociación acquaintance no implica que los objetos intercambien información sino que una instancia del objeto conoce de la existencia de la otra. La cardinalidad se utiliza para expresar el hecho de que un objeto puede estar asociado con varias instancias de otra clase.



Un caso especial de asociación acquaintance es la asociación *consiste-de* que se usa para establecer que un objeto se compone de otros formando un *aggregate*.

El *objeto entidad* modela información en el sistema que debe ser mantenida por un tiempo largo. Típicamente modela información que sobrevive a los casos de uso. Todo el comportamiento asociado a esa información debe ser puesto en el objeto entidad. Usualmente los objetos entidad se corresponden con entidades de la vida real, fuera del sistema

Para almacenar la información los objetos utilizan *atributos*. Cada objeto entidad puede tener varios atributos. Cada atributo tiene tipo. Un atributo es descripto como una asociación con nombre y cardinalidad indicando el tipo del atributo.



Una operación sobre un objeto entidad puede provocar que el objeto contacte otro objeto entidad. Esta comunicación tiene lugar a través de la asociación *communication*. Una asociación communication modela la comunicación entre dos objetos. A través de esa asociación un objeto envía y recibe estímulos.

El *objeto control* modela funcionalidad que no es adjudicable a ninguno de los tipos de objetos anteriores. Por ejemplo, comportamiento que consiste en operar con varios objetos entidad, hacer algunos cálculos y devolver el resultado a un objeto interfaz.

En el modelo de requerimientos se especifica la funcionalidad completa del sistema. Podríamos decir que un caso de uso es particionado en objetos análisis. En la práctica esto significa, que la funcionalidad especificada en un caso de uso, debe ser distribuida entre varios objetos.

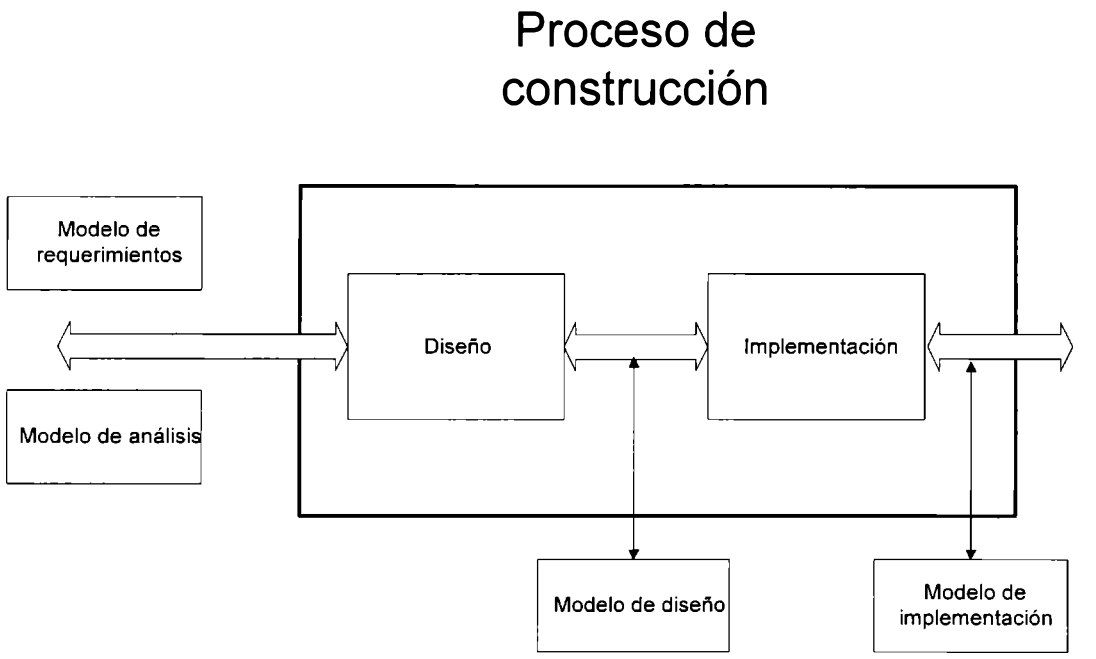
Básicamente la partición del caso de uso se hace de acuerdo a los siguientes principios:

- Toda aquella funcionalidad del caso de uso que es directamente dependiente del ambiente del sistema se ubica en objetos interfaz.
- Toda aquella información que tiene que ver con el almacenamiento y manejo de información que no tiene que ver con la interfaz, es puesta en objetos entidad.
- La funcionalidad a uno o pocos casos de uso y que no se ubica naturalmente en otros objetos se ubica en objetos control.

Una vez que los objetos son identificados y especificados, también se describe cómo esos objetos están relacionados, para eso se utilizan relaciones de distintos tipos. A menudo es recomendable hacer un Use Case View, que es una vista de los objetos, que muestra cómo se combinan para hacer el caso de uso, acompañada de una descripción del caso de uso en términos de los objetos identificados. Esto debe colaborar en hacer una estructura más robusta y mantenible.

Subsistemas: Los subsistemas son grupos de Objetos cuya tarea es reducir la complejidad. Un sistema se compone de varios subsistemas los cuales a su vez contienen subsistemas y así seguimos recursivamente hasta encontrar, al final de la jerarquía, los objetos análisis.

Modelo de diseño



El modelo de diseño es un refinamiento y formalización del modelo de análisis. Las razones principales para la construcción de este modelo son:

- El objetivo es refinar y formalizar lo suficiente el modelo de análisis para que sea fácil escribir el código a partir de él. Se deben describir las operaciones que cada objeto debe ofrecer, como interactúan los objetos, etc.
- El trabajo inicial cuando se desarrolla el modelo de diseño es adaptarlo al ambiente de implementación. Se deben considerar, por ejemplo, requerimientos de performance, lenguajes de programación, manejadores de bases de datos, etc.
- Validar los resultados del análisis. Construyendo este modelo se ayuda a descubrir puntos oscuros en el análisis en forma temprana, permitiendo volver a la etapa de análisis y corregirlos a costo razonable.

Las estructuras con las que se trabaja son básicamente las mismas que para el modelo de análisis. Sin embargo el punto de vista cambia ya que se está un paso más cerca de la implementación. Para Jacobson es importante tener en cuenta que las decisiones tomadas en esta etapa deben influenciar lo menos posible la estructura desarrollada en la etapa de análisis.

Las tareas desarrolladas en el proceso de construcción son:

- Identificar el ambiente de implementación. Este paso incluye la identificación e investigación de las consecuencias que el ambiente de implementación tiene en el diseño.
- Incorporar las conclusiones de la tarea de identificación y desarrollar una primera aproximación al modelo de diseño.
- Describir cómo los objetos interactúan en cada caso de uso específico.

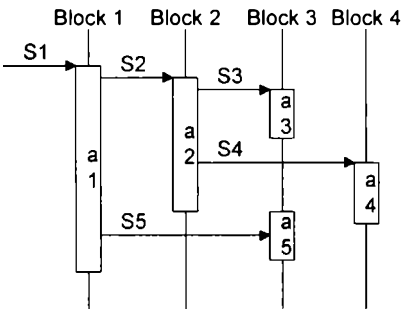
Para el desarrollo del modelo de diseño se introduce el concepto de *Bloque* para describir la intención de cómo el código debería ser producido (Clases).

El primer paso hacia el modelo de análisis puede hacerse mecánicamente, basados en el modelo de análisis. Inicialmente cada objeto de análisis se convierte en un Bloque. Luego, y teniendo en cuenta varias reglas y heurísticas propuestas por la metodología, se introducen cambios al modelo inicial de varias maneras:

- Introduciendo nuevos bloques que no tienen representación en el modelo de análisis.
- Borrando bloques.
- Cambiando bloques (Splitting y Joining de bloques existentes).
- Cambiando las asociaciones entre bloques en el modelo de diseño.

Una vez obtenidos los bloques, la tarea siguiente es refinarlos. La forma de refinar bloques para acercarnos al código es describiendo cómo se comunican en ejecución. Para describir la comunicación entre bloques se usan Estímulos. Un estímulo es enviado de un bloque a otro para disparar una ejecución en dicho bloque. La ejecución puede enviar estímulos a otros bloques.

Para describir la secuencia de estímulos se utilizan *diagramas de interacción.*



El diagrama de interacción muestra cómo los objetos participantes realizan un caso de uso a través de su interacción. La interacción es representada como bloques mandándose estímulos unos a otros. Los diagramas de interacción son controlados por eventos. Un evento da lugar a una operación. Los eventos son estímulos que se envían de un objeto a otro e inician una operación.

Un estímulo puede tener diferente semántica según sea intraproceso (Mensaje) o sea interproceso (Signal). Estos últimos pueden se sincrónicos o asincrónicos (la ejecución del enviador continúa inmediatamente después del envío del Signal).

El principal objetivo de los diagramas de interacción es describir los protocolos de los bloques. Si recorremos todos los diagramas de interacción obtenemos una interfaz

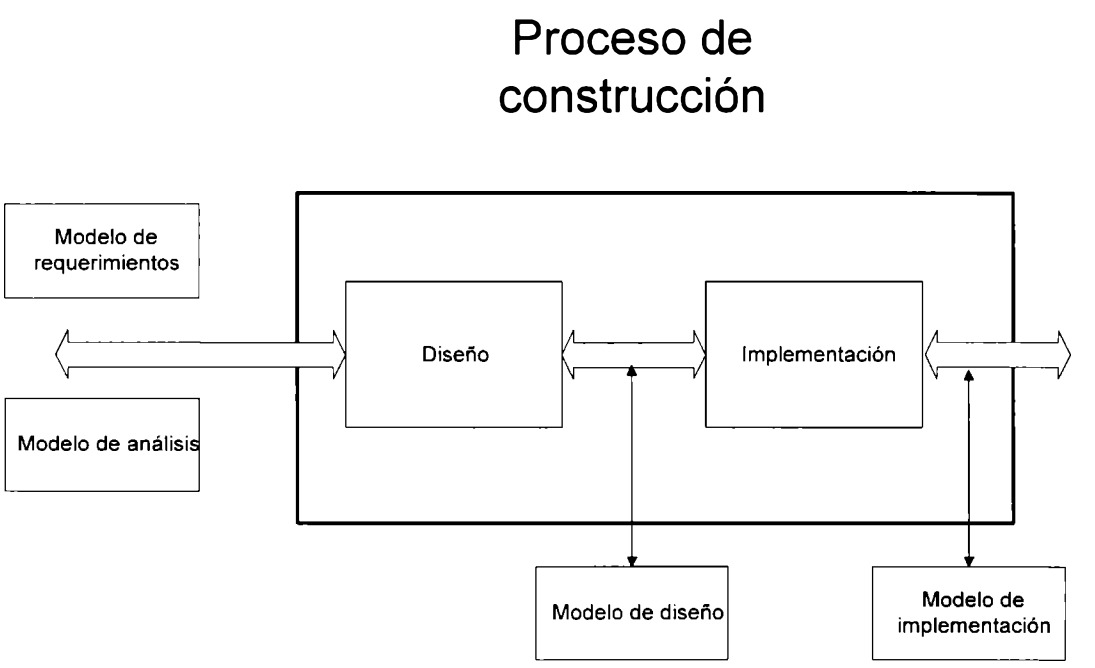
completa del bloque. De esta manera es posible congelar la interfaz de los bloques y comenzar su diseño en paralelo.

Un paso intermedio antes la implementación real puede ser un diagrama de transición cuyo propósito es obtener una descripción simplificada del bloque sin recurrir al lenguaje de programación.

Un problema durante el proceso de construcción es que la complejidad aumenta enormemente, para manejar el sistema más abstractamente se introduce el concepto de subsistema. Un subsistema agrupa varios objetos. Pueden ser usados tanto en el modelo de análisis como en el modelo de diseño. Los subsistemas pueden incluir otros subsistemas. El nivel más alto es el de sistema.

Cuando se trabaja manteniendo y desarrollando nuevas versiones de un producto, es apropiado a menudo extraer las interfaces al nivel de los subsistemas. De esta manera, no es necesario para cada equipo de desarrollo conocer la estructura interna de cada subsistema.

Modelo de implementación

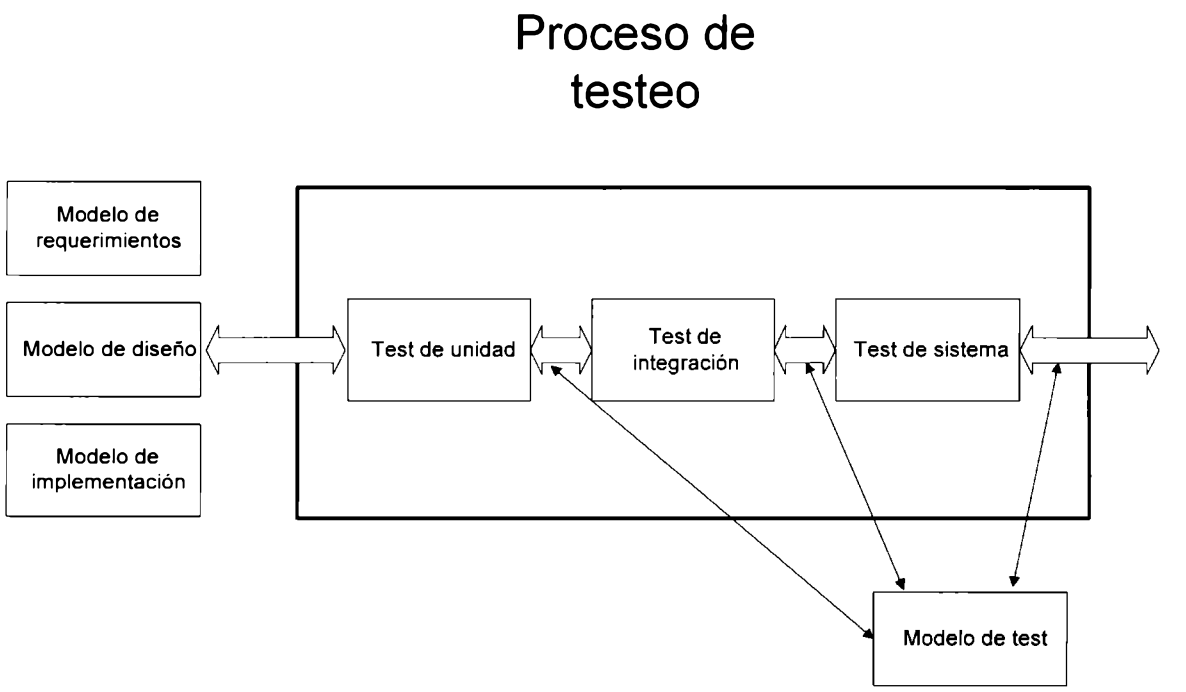


El modelo de implementación consiste del código fuente. Jacobson no requiere de un leguaje de programación orientado a objetos. La técnica puede ser usada con cualquier lenguaje de programación, sin embargo es deseable un lenguaje orientado a objetos dado que fácilmente los conceptos fundamentales pueden ser mapeados directamente a constructores del lenguaje.

La base para el modelo de implementación es el modelo de diseño. En él se ha especificado la interfaz de cada bloque y el comportamiento que se espera de esa interfaz. Típicamente un bloque mapea a 1-5 clases.

La posibilidad de usar componentes es un herramienta extremadamente potente. Las componentes son partes completamente implementadas.

Modelo de testeo



El modelo de testeo es el último modelo desarrollado en el desarrollo del sistema. Los conceptos fundamentales son la *especificación de test* y el *resultado del test*.

Inicialmente se testean los niveles inferiores como los bloques, luego se testean los niveles de subsistemas más bajos. De esta manera se asciende por los niveles con el objetivo de testear los casos de uso y finalmente todo el sistema.

El modelo de requerimientos también forma una herramienta potente. Al testear casos de uso, chequeamos que los objetos se comunican correctamente. Similarmente se chequea que las interfaces descriptas en el modelo de requerimientos funcionen correctamente. De esta manera logramos que el modelo de requerimientos sea verificado en el modelo de testeo.

Herramientas CASE

CASE (Computer-Aided Software/System Engineering) es "primariamente la integración de tecnologías orientada a la producción para mejorar el desarrollo de software y sistemas" (Chikofky 1988). Las herramientas CASE se aplican bien para modelar interfaces humanas y de hardware.

Existen diferente clases de herramientas para soportar los diferentes subprocesos, métodos, técnicas y modelos de proceso del desarrollo de sistemas de información. Básicamente las herramientas CASE permiten modelar y documentar los sistemas de información desde los requerimientos originales a través del diseño hasta la implementación (Chikofky & Rubenstein 1988).

Hay sistemas de diseño enfatizan las etapas tempranas del desarrollo tales como planeamiento, análisis y diseño. Tales herramientas son conocidas como "Upper CASE systems". Por otro lado los sistemas de implementación o "Lower CASE systems" enfatizan las actividades tardías como la programación, mantenimiento y reuso. (Oinas 97)

Requerimientos de herramientas CASE

La rápida evolución de las metodologías y la tecnología agregan permanentemente requerimientos a las herramientas CASE, lo que exige de ellas una gran *adaptabilidad* y permanente reformulación, presentando grandes desafíos para la investigación. Algunos de requerimientos mas importantes se enuncian a continuación:

Un CASE debería *mantener, simultáneamente, la información de varios desarrollos* de soft. Podemos enunciar dos razones de importancia para ello:

La primera es que en general un usuario de un CASE lleva adelante mas de un proyecto a la vez. Para este problema existen soluciones alternativas que no implican la necesidad de mantener varios desarrollos. Por ejemplo, este problema podría ser solucionado proveyendo distintas "imágenes" de la herramienta CASE , una para cada proyecto.

La segunda razón es determinante: la necesidad de re-usar los esfuerzos de desarrollos previos determina que la información de otros proyectos deba estar muy accesible y pueda ser compartida.

La necesidad de reusar esfuerzos previos no solo exige que se mantenga la información de varios proyectos. El reuso también exige que el CASE provea *múltiples formas de acceso a la información* de los proyectos, de manera de poder utilizar criterios flexibles y poderosos para encontrar los artefactos reusables que han sido desarrollados en proyectos previos.

Una herramienta CASE debe incluir un *modelo de la forma organizacional del proceso de desarrollo*. La modalidad de desarrollar en grupos presenta un desafío a las herramientas CASE, el de dar soporte a la actividad de varios usuarios y brindar a cada uno una visión de los desarrollos acorde con el rol que desempeña dentro del equipo.

Las herramientas CASE deben dar *soporte a formas poco estructurada de generar y relacionar información*. Si bien las metodologías tratan que los desarrollos

sean hechos de una forma ordenada, la poca estructuración de la información es intrínseca de esta actividad, sobre todo en las primeras etapas (Análisis y diseño).

El mantenimiento de la *consistencia* entre los documentos generados por la herramienta CASE es un requerimiento fundamental y es tema central en un gran número de trabajos.

Las metodologías incluyen un número de *lenguajes y técnicas de descripción* que son utilizadas a lo largo del desarrollo, es necesario que una herramienta CASE soporte en forma eficiente estas estructuras conceptuales y que además lo haga en forma integrada de manera de asegurar la calidad del producto.

Al igual que las metodologías de desarrollo, una herramienta CASE, debe soportar varios *niveles de abstracción* de manera de auxiliar al desarrollador para manejar la complejidad intrínseca de los problemas.

Una herramienta CASE debe cubrir el *espectro completo de las actividades del desarrollo* de sistemas de información. (El ciclo de vida Completo). Deberá incluir un modelo o patrón de las actividades a cumplir durante el proceso de desarrollo. Es muy común encontrar recorridos por defecto de la información contenida en el repositorio y ayudas al usuario que le faciliten la aplicación de la metodología.

Un requerimiento muy frecuente es el de mantener consistente y accesibles distintas *versiones* de los artefactos de software, manejándolas de manera tal que el usuario pueda introducir modificaciones sin afectar el resto de los desarrollos, logrando un grado mas en el reuso de otros esfuerzos.

Otros requerimientos que se presentan frecuentemente son: *Interfaz gráfica ,soporte de varias tareas concurrentes, generación automática de código, soporte para procesos de reingenieria de software.*

Lyytinen, Smolander, Tahvanainen han caracterizado un ambiente CASE verdadero definiendo las diferentes características necesarias y suficientes (1 y 2) y adicionales (3 a 6):

1. Debe incorporar varias estructuras conceptuales y lenguajes de descripción.
2. Debe soportar varios niveles de abstracción en los cuales el proceso de desarrollo tiene lugar.
3. Debería cubrir espectro completo de actividades en el desarrollo de un sistema de información.
4. Debería incluir un modelo de un patrón necesario de actividades a ser llevadas a cabo durante el proceso de desarrollo.
5. Debería incluir un modelo de la forma organizacional del proceso de desarrollo.(Roles humanos) y
6. Debería utilizar descripciones e implementaciones existentes.

Características comunes a Herramientas CASE

Podemos enumerar un conjunto de características que, llevadas al dominio de las herramientas CASE, ayudan a satisfacer los requerimientos enunciados en la sección anterior. Algunas de ellas se detallan a continuación:

Soporte para grupos de desarrollo.

- Soporte para distintos roles de usuarios.
- Derechos de acceso (Soporte para estructuras de grupos de trabajo).
- Ejecución multitarea.
- Ejecución multiusuario.

Manejo de proyectos.

Manejo de versiones.

Manejo de alternativas.

Chequeo de consistencia.

Chequeo de completitud.

Manejo de bibliotecas de componentes.

Distintas visiones de la misma información.

- Reportes.
- Gráficos.
- Texto.
- Tablas.
- Otros medios.

Querys.

Editores gráficos de documentos.

Recorrido por defecto de los documentos del sistema.

Descomposición de objetos complejos.

- Herramientas de navegación.
- Mapas.

El requerimiento de Reuso

El reuso es una de las estrategias principales para lograr incrementar la productividad de los desarrolladores de software. Para lograr la construcción de una herramienta CASE exitosa deben considerarse críticas las facilidades de reuso que provea.

La posibilidad de reuso depende en gran medida de la existencia de una librería de artefactos reusables lo suficientemente extensa, pero hasta la mejor librería pierde toda utilidad si no se posee herramientas y métodos de búsqueda lo suficientemente eficientes, la interfaz hipermedial provee solución a este problema.

La especificación de las herramientas de interfaz hipermedial (Browsers), la estructura de los documentos en el repositorio y las relaciones entre los documentos requieren un estudio detallado de las formas mas usuales de localización de los candidatos a ser reusados y de la forma en que estas estrategias de acceso son provistas por un repositorio con interfaz hipermedial.

La Interfaz hipermedial permite al usuario una gran flexibilidad en la búsqueda ya que le permite abstraerse de los nombres o especificaciones exactas de los artefactos y realizar la búsqueda navegando por relaciones (links) que se adaptan mucho mejor a la estructura de pensamiento de un desarrollador, de esta manera la hipermedia lo ayuda a descubrir ítems aptos para el reuso. Incluso el usuario inexperto puede moverse por relaciones predefinidas basadas en los atributos mas comunes de los documentos que es mucho mas natural que otras aproximaciones como los querys (aunque no puede despreciarse su utilidad).

Un browser hipermedial debe proveer múltiples formas de navegación y de búsqueda en el repositorio, lo cual determina la consideración de una variedad de descriptores en los documentos y relaciones del repositorio. "Pattern matching" exacto en todos los descriptores, clasificación jerárquica facetada, relaciones libres o combinaciones de los anteriores son formas de búsqueda sumamente útiles.

La búsqueda de un artefacto de software en un repositorio es un ejemplo excelente para la aplicación de los conceptos de interfaz hipermedial, ver la definición de Harry

Al diseñar la herramienta basada en un Repositorio se pretende alentar la construcción, con bajo costo, de herramientas potentes de búsqueda y clasificación de elementos.

El requerimiento de adaptabilidad y extensibilidad

El enorme crecimiento en el tamaño del software y la confiabilidad que se requiere de él ha hecho que se centre la atención sobre las metodologías, lenguajes y correspondientes herramientas. Las herramientas no sólo soportan la programación sino que también soportan la especificación, el diseño y las demás etapas del ciclo de vida de un artefacto de software, incluyendo la documentación. Esta forma de trabajo genera una gran cantidad y variedad de documentos que deben ser guardados, organizados y accedidos en forma eficiente.

Es común encontrar que las herramientas CASE soporten todas las fases del ciclo de vida, pero ajustándose a cierta metodología de desarrollo. Nosotros creemos que hay dos características que hacen que una herramienta de este tipo tenga desventajas:

Existe un gran número de metodologías de desarrollo, con diferencias considerables entre ellas, y *ninguna ha sido adoptada como norma general.* Al soportar una sola metodología estamos brindando apoyo a un numero reducido de usuarios, los que utilizan esa metodología. Además debemos descartar la posibilidad de tener un repositorio con varios proyectos integrados en el caso que fueran desarrollados con metodologías diferentes a la que soporta la herramienta. De esto se desprende una notable disminución en las chances de reuso.

Las metodologías están en permanente evolución. La permanente evolución de las metodologías produce una constante desactualización y problemas de compatibilidad, causando efectos parecidos a los que se describen en el punto anterior (Aunque por razones distintas).

Por tales motivo pensamos que la adaptabilidad a los nuevos requerimientos y la capacidad de dar soporte a distintas metodologías de trabajo son esenciales para una herramienta CASE y creemos que un diseño adecuado y la utilización de tecnología de objetos e interfaz hipermedial son de suma utilidad en este sentido.

Hipermedia

La idea de hipertexto es permitir la fácil combinación y uso de unidades de información por parte de usuarios humanos. Básicamente hipertexto empaqueta información en unidades pequeñas llamadas nodos, los cuales están conectados unos con otros por links.

Un *nodo* es un contenedor de información el cual provee una representación para estructurar elementos de información.

Un *link* especifica la conexión semántica entre dos nodos.

Un *ancla* designa la fuente o destino de un link, pudiendo ser un nodo o parte de un nodo.

Un *hiperdocumento* es una colección de nodos con sus respectivos links.

Un *hiperespacio* es una colección de hiperdocumentos y sus links. Hiperespacios colaborativos son hiperespacios compartidos por personas en un equipo o una organización.

Los *autores* son los que crean las componentes de hipertexto y los *lectores* son los que "navegan" a través de la información.

Se dice que hipertexto se puede aplicar a una amplia gama de dominios de aplicación, por ejemplo Ayudas en línea, correo electrónico, sistemas operativos, manejo de proyectos, modelización financiera, publicidad electrónica e ingeniería de software. Han aparecido sistemas de hipertexto muy populares como World Wide Web (Berners-Lee et al.) o HiperCard (Goodman 1990).

Hipertexto no se puede usar en todos los sistemas de información, Shneiderman y Kearsley (1989) han definido las "Tres reglas de oro" para decidir cuando usar presentación hipertextual:

1. Hay mucha información
2. La información esta compuesta naturalmente en piezas pequeñas
3. La relación entre las piezas puede ser definida y representada.

Los desarrollos de software generan un espacio de información que satisface las tres reglas, por lo tanto podemos considerar que son apropiados para presentarlos hipertextualmente.

En varios artículos se ha discutido problemas potenciales de la tecnología de hipertexto. (Raskin 1987, Halasz 1988, Halasz 1991). Dos problemas primarios son tendencia a perder el sentido de ubicación y dirección en un documento no lineal y esfuerzo adicional de concentración necesario para mantener varias tareas o " caminos" a la vez (Conklin 1987).

El problema de desorientación implica que los usuarios no saben donde están en el hiperespacio y no saben como ir a otro lugar. Las dos soluciones para manejar este

problema son los browsers gráficos y los mecanismos query/search (Conklin 1987). Los ambientes espaciales creados a través de browsers gráficos ayuda a los usuarios a orientarse a través de patrones visuales. Las técnicas de búsqueda para localizar nodos y links por String completo, palabras clave o predicados lógicos (autor, momento de creación, tipo, etc.) tanto como los filtros y vistas diferentes de la misma información puede ayudar a la orientación.

La sobrecarga cognitiva es la dificultad a acostumbrarse a la creación, asignación de nombres y seguimiento seguimiento de links (Conklin 1987). El autor del link tiene que decidir que señal quiere asociar a cada link (por ejemplo tipo y label) y debe considerar la creación de suficiente cantidad de links. La autoría fragmentada de numerosos nodos al mismo tiempo puede traer problemas como la segmentación de ideas prematura.

El lector se encuentra con una cierta sobrecarga de decisiones, por ejemplo si tiene una gran cantidad de links para seguir o si los labels de los links son suficientemente informativos. Sin embargo este problema puede verse como una clase diferente de problema mas que como un problema adicional dado que se pueden dar síntomas similares con texto lineal.

HCase

Descripción General

Luego de relevar los requerimientos mas importantes de las herramientas CASE comenzamos a analizar y diseñar HCase, si bien sabíamos que no íbamos a poder cubrir muchos de los requerimientos, nos propusimos hacer un diseño lo suficientemente general para que la tarea de cubrirlos se mediante extensiones y no alteraciones al diseño.

Para analizar HCase ideamos una estrategia

- Observar el CASE desde distintos puntos de vista
- Desde cada punto de vista construir un modelo
- Analizar cada modelo y a partir de los resultados obtener:

  Nuevos requerimientos o refinamiento de los requerimientos originales.

  Especificación clara de la funcionalidad.

  Alternativas de diseño.

Los modelos analizados fueron:

Un modelo de metodología de desarrollo

Un modelo al que se pueda aplicar teoría de grafos

Un modelo al que se pueda analizar con lógica de primer orden

Un modelo hipermedial

Un modelo de bases de datos.

Un repositorio de documentos accedidos por un conjunto de herramientas integradas.

Una herramienta CASE

De la construcción de estos modelos surgió un conjunto de conclusiones que sirven como especificación de requerimientos y diseño preliminar de HCase. En la sección donde se describe en detalle el diseño de HCase se podrá observar la influencia de cada modelo.

Metodología de desarrollo

Básicamente las herramientas CASE permiten documentar y modelar los sistemas de información desde los requerimientos originales a través del diseño hasta la implementación (Chikofky & Rubenstein 1988).

HCase modela una metodología de desarrollo, el modelo es construido en función de las siguientes ideas:

- La aplicación de una metodología genera un conjunto de documentos.

- Los documentos que se generan cuando se aplica una metodología son interdependientes.
- Podemos modelar una metodología modelando los documentos que esta genera y la forma en que se asocian

Las ideas anteriores sugieren que el HCase tome la forma de un conjunto de documentos que guarden la información de distintos proyectos. Estos documentos no solo deben guardar la información sino que deben administrar su corrección. Además, HCase debe conocer y aplicar el conjunto de reglas y dependencias que existe entre los distintos documentos. Como funcionalidad adicional HCase debe proveer formas de buscar, clasificar, actualizar y acceder a la información de los documentos.

Documento: Maquina recicladora

Tipo: Proyecto
Proyecto: Maquina recicladora
Autor: Darío y Gustavo
Fecha: 9/12/97
Descripción: El sistema controla una máquina de reciclado para botell retornables, latas y cajas. La máquina puede distinguir q clase de botella o lata esta reciclando. La maquina tambi emite un recibo para el cliente con un detalle de lo reciclado puede llamar a un operador cuando un ítem esta atascad además emite un detalle de lo recibido en el día.

Compuesta por: Modelo de CU
Compuesta por: Modelo de Diseño
Compuesta por: Modelo de análisis

Revision1: 10/12/97 Gustavo
Revision2: 11/12/97 Darío

Documento: Modelo de análisis

Tipo: Modelo de analisis
Proyecto: Maquina de reciclado
Autor: Dario y Gustavo
Fecha: 9/12/97
Descripción: El modelo de analisisdescribe los casos de uso en términos de objetos interfaz, entidid y control.

Traceability desde: Modelo de Casos de Uso
Traceability hacia: Modelo de Diseño
Contiene OI: Panel del Cliente
Contiene OI: Impresora de Recibos
Contiene OC: Receptor del ítem depositado
Contiene OE: Recibo
Contiene OE: Item depositado
Contiene OE: Lata
Contiene OE: Botella
Contiene OE: Caja
Graficado en Gráfico de MA

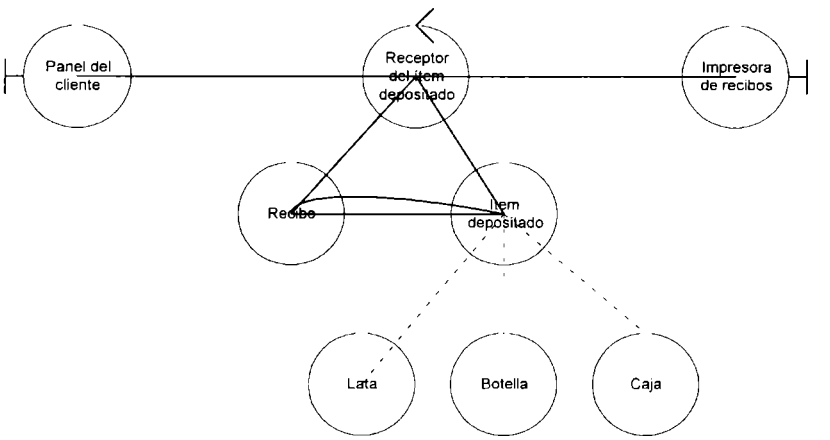
Documento: Item depositado

Tipo: Objeto entidad
Proyecto: Máquina de reciclado
Autor: Darío y Gustavo
Fecha: 9/12/97
Descripción: El objeto modela cualquiera de los objetos que pueden ser recibidos por la máquina recicladora, botellas, latas o cajas.

Item de: Recibo
Superclase de: Lata
Superclase de: Botella
Superclase de: Caja
Conoce A: Receptor de ítem depositado

Documento: Modelo de análisis

Tipo: Objeto entidad
Proyecto: Máquina de reciclado
Autor: Darío y Gustavo
Fecha: 9/12/97
Descripción: El objeto modela cualquiera de los objetos que pueden ser recibidos por la máquina recicladora, botellas, latas o cajas.



Desde este punto de vista podemos sacar algunas conclusiones relevantes:

Los Documentos tienen contenido

El contenido modeliza la información perceptible del documento.

El contenido tiene un aspecto sintáctico y otro semántico. En el aspecto sintáctico generalmente se define para cada documento un conjunto de figuras o símbolos que pueden aparecer en el contenido y un conjunto de reglas sintácticas que deben ser cumplidas. El aspecto semántico en general es una descripción de la forma en que los elementos sintácticos modelan entidades, la corrección semántica es muy difícil de testear y las reglas semánticas, en general no están rigurosamente definidas.

Aunque generalmente es textual o gráfico, no hay restricciones acerca del contenido de un documento, puede ser audio, video, etc. Un caso muy interesante es el de un grafo de colaboraciones donde aparecen varias tarjetas CRC, en este caso el contenido es una agregación de documentos, (subconjunto de documentos asociados con características propias, independientes del documento contenedor).

Los documentos tienen descriptores

Cada documento posee un conjunto de atributos que describen algunos de sus aspectos y propiedades, algunos ejemplos son Fecha de creación, nombre de los autores, nombre del proyecto, nombre del documento, dominio de aplicación, etapa del ciclo de vida, fecha de ultima modificación, palabras clave, etc.

Estos atributos definidos como descriptores no son parte del contenido del documento y son generalmente independientes del contenido del documento

Los descriptores son utilizados por herramientas para implementar criterios de búsqueda, clasificación de documentos y estrategias de rehuso, un usuario puede buscar un documento por autor, fecha de creación o proyecto.

Los documentos están asociados con otros documentos

En los documentos podemos hacer que algunos elementos sintácticos (figuras, frases, etc.) hagan referencia a otras partes del documento u otros documentos, muchas de estas referencias están estipuladas en la metodología (Tarjeta de clase y superclase) y otras son definidas por el usuario (Nota al pie).

Hay asociaciones que son internas al documento (nota al pie) y hay otras que van desde un documento a otro (un CU "Conoce a" otro CU).

A diferencia de los descriptores, las asociaciones no siempre son independientes del contenido, es decir que si cambia el contenido pueden cambiar las asociaciones con los otros documentos, un ejemplo típico se da con un Caso de uso cuya descripción textual hace referencia a otro documento que describe un Actor, en ese caso, un cambio en el texto podría hacer desaparecer la asociación.

Los descriptores y las asociaciones son generalmente independientes.

Existen distintas formas de relacionar documentos, distintos tipos de asociaciones, y HCase debe tener la capacidad de conocer las propiedades de cada una y administrar la correcta utilización de las mismas.

Existen distintos tipos de documentos

En general las metodologías describen distintos tipos de documentos. El tipo no es un descriptor porque no es independiente del contenido, aunque puede ser utilizado como uno. El tipo determina:

El aspecto del documento al ser percibido por un usuario.

El aspecto sintáctico y semántico del contenido del documento.

Los criterios que deben aplicarse para determinar la corrección del contenido.

Los tipos de asociaciones que acepta el documento.

El conjunto de documentos con los que puede estar asociado.

Los criterios que deben aplicarse para determinar la corrección de las asociaciones.

El conjunto de descriptores que tendrá, (Aunque muchos son comunes a todos los documentos).

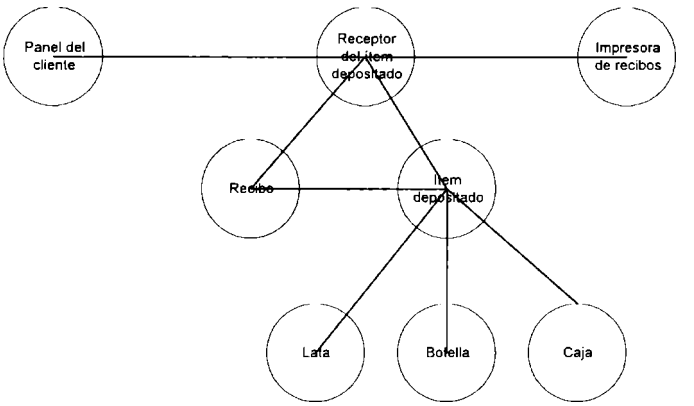
El conjunto de valores que pueden tomar los descriptores.

Grafo con arcos y nodos

Otro de los puntos de vista con el que podemos analizar a HCase es el de un grafo con arcos y nodos, una definición seria como sigue:

Cada documento es un nodo del grafo.

Cada asociación representa un arco.



En base a la definición anterior tratemos de observar los documentos de la metodología desde este nuevo punto de vista:

Hay distintos tipos de nodos.

Hay distintos tipos de arcos.

Cada nodo posee un contenido.

Cada nodo posee un conjunto de descriptores.

Cada nodo tiene un tipo.

El tipo del nodo determina:

El aspecto del nodo al ser percibido por un usuario.

El aspecto sintáctico y semántico del contenido del nodo.

Los criterios que deben aplicarse para determinar la corrección del contenido.

Los tipos de arcos que acepta el documento.

El conjunto de nodos entre los que puede haber arcos.

Los criterios que deben aplicarse para determinar la corrección del grafo.

El conjunto de descriptores que tendrá, (Aunque muchos son comunes a todos los documentos).

El conjunto de valores que pueden tomar los descriptores.

Si bien es cierto que este punto de vista no agrega información con respecto al interior de un nodo (Es igual al interior del documento), también es cierto que presenta notables cambios con respecto a la forma en que se presentan las asociaciones. En el punto de vista de la metodología las asociaciones aparecen como propiedades del documento, en este nuevo punto de vista las asociaciones (Arcos) son objetos individuales y existen en forma separada a los nodos.

El nuevo punto de vista aporta varias características positivas:

Definición clara de requerimientos

Los arcos son objetos y existen independientemente de los documentos. Esto es importante a nivel de diseño ya que alienta la creación de un nueva clase "Arco" con responsabilidades y atributos propios.

Los arcos conectan exactamente dos nodos (Binarias).

Los arcos tienen un origen y un destino (Dirigidos).

Los arcos tienen descriptores.

Los descriptores de documentos no poseen arcos.

El contenido de un nodo puede tener arcos, estos se consideran como arcos del nodo, por ejemplo una descripción textual con arcos a otros nodos.

Solución de chequeos de consistencia usando herramientas matemáticas

Teniendo como base un grafo es posible solucionar fácilmente muchos chequeos de consistencia, algunos ejemplos de ellos son.

"Un arco tipo "Extends" puede conectar solo casos de uso". En general: Para cada tipo de arco puedo definir los tipos de nodos puede conectar.

"Todo Caso de uso debe tener un Actor iniciador", "Herencia múltiple". En general: Determinar la cantidad máxima y mínima, si las hay, de arcos de determinado tipo que pueden entrar o salir de un nodo.

"Una Clase no puede heredar de si misma". En general: Determinar la existencia de ciclos (Caso del circulo y elipse).

Excelente base para mapas de hipermedia

El grafo es una forma tradicional de hacer mapas de hipermedia, existen muchos trabajos que estudian la visualización de grafos que pueden ser útiles para este fin.

Utilización de propiedades de grafos para hacer browsers intuitivos

Podemos utilizar propiedades de los grafos para especificar fórmulas que calculen la distancia entre dos nodos, si esta distancia refleja la probabilidad de que un usuario desee navegar a ese nodo podemos utilizarla para hacer browsers muy potentes que muestren solo los nodos mas "Cercanos" (Afinity browser).

Planteo claro de problemas

Podemos usar terminología de grafos para especificar claramente problemas, soluciones y propiedades. Por ejemplo veamos un nodo que modela un grafo de colaboraciones, en este caso se ve claramente que el contenido del nodo es un subgrafo con nodos (Tarjetas) y arcos ("Colabora con") propios.

Visión desde nuevas perspectivas

Utilizando este punto de vista aparecen claramente conceptos que desde otro punto de vista no son tan claros, por ejemplo desde el punto de vista de la metodología es difícil explicar la diferencia entre asociación y descriptor: Ambos son atributos de los documentos (Nodos) pero los descriptores no son arcos del grafo (no asocian documentos).

Predicados aplicados a un dominio

Este punto de vista surge a partir del punto de vista del grafo, haciendo un modelo mas abstracto del repositorio, una definición seria:

Los nodos son elementos de un dominio

Los arcos como relaciones binarias entre los elementos del dominio

Utilizando este modelo podemos establecer propiedades y requerimientos de HCase a través de predicados lógicos. Este nuevo punto de vista aporta un lenguaje formal para expresar reglas de consistencia claras y rigurosas, con la ventaja de facilitar la traducción de las reglas a algoritmos implementables.

Algunos ejemplos de reglas son:

- Para todo a, b Tarjetas de clase (Si a "HeredaDe" b y b "HeredaDe" c => a "HeredaDe" c)
- Para todo a, b Documentos (Si a "TraceabiityA" c => no c "TraceabilityA" a)
- Para todo a, b Documentos (Si a es "ParteDe" b => b no puede ser "ParteDe" a)
- "ParteDe" es transitiva.
- Todo caso de uso debe tener un actor iniciador
- Si CU1 es iniciado por A1 y CU1 pertenece al modelo M1 entonces A1 pertenece aM1

Espacio hipermedial

HCase se puede ver como una red hipermedial en la cual un browser adecuado puede permitir navegar entre los documentos utilizándolos como nodos. Para lograr este objetivo debemos tener un modelo de la hipermedia, el cual describimos a continuación:

Un *nodo* es un Documento.

Un *link* es una Asociación o un link definido a nivel de interfaz.

Un *ancla* depende de la presentación.

Un *hiperdocumento* es un Proyecto.

El *hiperespacio* es HCase.

Los *autores* y los *lectores* son los desarrolladores.

Dado que los documentos son los nodos es posible construir browsers que permitan navegar por los distintos documentos.

Pero mas importante es que las distintas asociaciones que existe entre los documentos funcionan como links. Esto tiene un impacto importantísimo en la herramienta, aportando ventajas determinantes.

Se brinda un soporte automático de generación de links, relevando al usuario de una tarea ardua como es la creación de links.

Como los liks son mapeados automáticamente cuando dos documentos esta relacionados de alguna manera, al navegar, tengo una alta probabilidad de encontrar el link que estoy buscando. Por ejemplo si estoy viendo el documento de una clase y quiero navegar al documento equivalente de su superclase, es muy probable que encuentre el link que me permita hacerlo. Si la creación estuviera a cargo del usuario podría no encontrarlo.

Consistencia. Dada la asignación biunívoca seguro que si existe la relación existe un link y viceversa.

Por otro lado aparecen algunas dificultades que deben ser resueltas en el marco de la herramienta.

Dado que cada asociación entre documentos es mapeada a un link, la cantidad de links puede ser muy importante, si la gran cantidad de links no es manejada con criterio puede entorpecer la tarea del usuario, por esto es necesario definir "Filtros" y clasificaciones para que el usuario perciba solo lo que necesita.

Existen distintas formas de navegar, mediante menúes, mediante barras de herramientas, mediante doble click o combinaciones de estas. Las herramientas hipermediales deben definir la forma de navegación seleccionada.

Existen varias posibilidades para hacer perceptibles los ancors, esto también depende de la forma en la que se presentan los nodos y del tipo de contenido de los mismos.

Las hipermedias poseen, además de la posibilidad de navegar, herramientas que ayudan al usuario a ubicarse en el espacio de navegación y a navegar fácilmente a la información que necesita. Es necesario definir un conjunto de herramientas adecuadas para esta función.

Base de datos

A veces es necesario obtener resúmenes y estadísticas que contemplen al conjunto de datos por completo.

Se puede modelar el repositorio de HCase como una base de datos donde los datos son valores de descriptores de documentos, su contenido y la información de las asociaciones entre ellos.

Proyecto	Nombre	Tipo	Descripción	Fecha	Autor
Máquina reciclador	Botella	Objeto entidad	Modela un item depositado	21/11/97	Gustavo
Máquina reciclador	Caja	Objeto entidad	Modela un item depositado	14/09/97	Darío y Gustavo
Máquina reciclador	Depositar ítem	Caso de Uso	El caso de uso comienza cuando el C	10/10/97	Darío
Máquina reciclador	Item depositado	Objeto entidad	Modela un item depositado	21/11/97	Darío y Gustavo
Máquina reciclador	Lata	Objeto entidad	Modela un item depositado	5/11/97	Darío y Gustavo
Máquina reciclador	Máquina reciclador	Proyecto	Es el proyecto de una máquina recicl	7/11/97	Darío y Gustavo
Máquina reciclador	Modelo de análisis	MA	Describe los casos de uso en término	21/11/97	Darío y Gustavo
Máquina reciclador	Modelo de CU	MCU	Modelo de Casos de Uso	21/11/97	Gustavo
Máquina reciclador	Modelo de diseño	MD	Modelo de Diseño	23/11/97	Darío y Gustavo
Máquina reciclador	Recibo	Objeto entidad	Modela un recibo	9/10/97	Darío

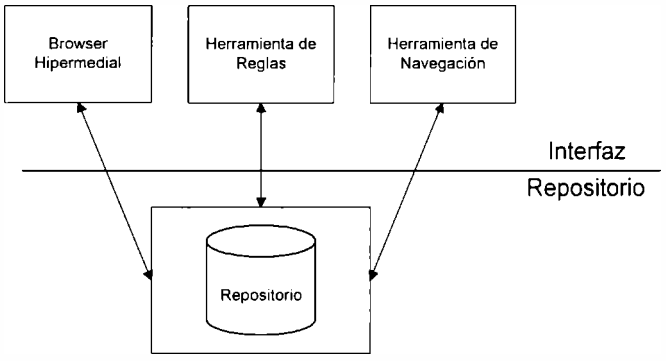
Visto de esta manera HCase es muy útil por ejemplo desde el punto de vista de las métricas o para hacer análisis del impacto de las modificaciones. Podemos sacar conclusiones como el promedio de documentos que genera cada autor, las desviaciones en la longitud de los documentos, la duración de un proyecto, el numero de proyectos en que participa cada usuario, la cantidad de veces que un documento es reusado etc.

Repositorio y herramientas integradas

Puede modelarse HCase dividido en dos niveles: Nivel de repositorio y nivel de interfaz.

A nivel de repositorio encontramos toda la documentación correspondiente a los desarrollos, por ejemplo descripciones de procesos, descripciones de interfaces, especificación de clases, gráficos, configuraciones de software y librerías de documentos en general.

A nivel de interfaz encontramos un conjunto de herramientas diseñadas para trabajar en base a los objetos del repositorio.



Esta es una decisión de suma importancia en el diseño de HCase. Usar un repositorio común, que maneje correctamente el modelo de objetos que soporta HCase, permite compartir información entre diferentes herramientas que soporten ese modelo de objetos.

La ventaja que se espera obtener con esta configuración es la de lograr un amplio *conjunto de herramientas altamente integradas* imputando en temas como:

Reuso

El repositorio puede verse como una librería de objetos reusable, desde este punto de vista permite clasificar los objetos y soporta herramientas que realicen querys y permitan explorar la biblioteca.

Permite hacer traking del uso de los objetos. Por ejemplo si un objeto es cambiado es fácil encontrar los proyectos que lo usan y actualizarlos.

El repositorio puede guardar información relacionada con el objeto reusable de manera de ser fácil de encontrar.

El repositorio puede reflejar dependencias entre varios objetos reusables.

Navegación

Como el repositorio maneja diferentes tipos de documentos y tenemos distintos tipos de herramientas mientras navegamos podemos abrir la herramienta apropiada al tipo de documento que estamos mostrando. Por ejemplo de una tarjeta de clase podemos saltar a un browser de Smalltalk para la clase o desde una descripción de interfaz podemos saltar al canvas.

Análisis de impacto

Cuando modificamos un objeto en el repositorio es fácil ver con que objetos esta relacionado, pudiendo hacer fácilmente un análisis preciso del impacto de la modificación.

Extensibilidad

El esquema de repositorio permite extender la funcionalidad de las herramientas y desarrollar nuevas independientemente de las otras, con la única restricción de respetar el modelo del repositorio, de esta manera se reduce la complejidad y el costo de esta tarea.

Modelos y requerimientos

De relevar los requerimientos generales de herramientas CASE y realizar el ejercicio de análisis y modelación, obtuvimos una conjunto de requerimientos que deben ser tenidos en cuenta y el impacto que el diseño de HCase tiene sobre ellos:

Brindar soporte automatizado a las actividades de la ingeniería de software

En el modelo de la metodología se evidencia la posibilidad de obtener soporte automatizado básico para la aplicación de una metodología, "HCase permite la generación y mantenimiento de un conjunto de documentos y asociaciones cada uno modelando y documentando un aspecto del desarrollo de software en cuestión".

En los otros modelos se presentan herramientas para el soporte de tareas de mayor nivel como búsquedas y reuso de desarrollos previos (Descriptores, Browsers hipermediales), coordinación, gestión y control de los desarrollos (Base de datos...), etc. Dado que también son requerimientos de las herramientas CASE veremos distintas formas de dar soporte mas adelante.

Adaptarse acorde a la rápida evolución de las metodologías y la tecnología

Este requerimiento pretende alcanzarse proveyendo un marco que permita la construcción de nuevas herramientas integradas y la extensión de la funcionalidad de las herramientas existentes. La base para este marco es la integración de HCase con la plataforma de desarrollo (Smalltalk), el diseño hecho como un conjunto de herramientas integradas basadas en un repositorio y la construcción basada en un Framework (HotDraw) .

Mantener, simultáneamente, la información de varios desarrollos de soft

Al modelar las metodologías mediante el conjunto de documentos que genera su aplicación y las asociaciones entre ellos y además almacenar la documentación y las asociaciones en un único repositorio es muy fácil satisfacer este requerimiento ya que la documentación de todos los proyectos convive en el mismo repositorio y puede ser accedida por las herramientas indiscriminadamente.

Permitir re-usar los esfuerzos de los otros desarrollos.

Al satisfacer el requerimiento de mantener simultáneamente la información de varios desarrollos estamos cumpliendo un requisito importante. Otro requisito es que el CASE provea múltiples formas de acceso y criterios poderosos para encontrar los artefactos reusables. La intefaz hipermedial, los mapas del grafo, las consultas por los descriptores o por predicados y los querys a la base de datos son herramientas suficientemente poderosas para satisfacerlo.

Soportar procesos de reingenieria de software, de manera de reusar descripciones y desarrollos ya existentes.

Hasta aquí explicamos como reusar desarrollos realizados dentro de la herramienta, existe la posibilidad de reusar desarrollos realizados con otras herramientas, dado que no hay restricciones acerca del contenido de los nodos y a herramienta permite la adaptación se pueden hacer que HCase soporte documentación realizada con otras herramientas.

Otra forma de reutilizar esfuerzos anteriores es incluir en HCase herramientas ya existentes como el Canvas o el Class Browser.

Incluir un modelo de la forma organizacional del proceso de desarrollo.

La combinación de la modelización mediante nodos y arcos (Documentos y asociaciones), la interfaz hipermedial que permite navegar por los arcos y la generación automática de nodos y arcos permite darle una forma organizacional al proceso de dearrollo, en el prototipo se eligió como criterio organizacional la asociación "Parte De", la cual presenta al desarrollo como un conjunto de documentos "Contenidos" unos adentro de otros a partir de un documento de proyecto. Existen otras posibilidades traceability, por autor, etc.

Dar soporte a formas poco estructurada de generar y relacionar información

Este requerimiento pretende ser cubierto con la posibilidad de reflejar las asociaciones asociaciones entre los distintos documentos que se encuentran sobre todo en las etapas de análisis y diseño.

Mantener la consistencia entre los documentos generados por el CASE.

En la posibilidad que permite el modelo de el repositorio y el conjunto de herramientas integradas de compartir el mismo documento en distintos desarrollos, sin hacer copias, esta la clave de la consistencia de HCase. Un buen diseño del repositorio permite a varios documentos compartir información común sin generar redundancia y manteniendo la consistencia en forma natural

Finalmente el punto de vista del grafo y de la satisfacción de predicados permite especificar e implementar el chequeo de varias reglas de consistencia.

Soportar, en forma integrada, el numero de lenguajes y técnicas de descripción que son utilizadas a lo largo del desarrollo.

La construcción del prototipo utilizando editores gráficos semánticos permite lograr una riqueza muy importante en lo que hace a lenguajes y técnicas de descripción. El concepto de independencia del contenido, la extensibilidad y la adaptabilidad permiten especular con una riqueza aun mas importante.

Soportar varios niveles de abstracción de manera de auxiliar al desarrollador para manejar la complejidad intrínseca de los problemas.

El repositorio posee una estructura organizacional básica que llamamos árbol de contenedores, según esta estructura los documentos que están en niveles de abstracción altos están cerca de la raíz del árbol, estos documentos "contienen" otros documentos que son visiones detalladas del documento abstracto. El ejemplo típico es un grafo de colaboraciones "contiene" tarjetas de clase. Esta estructura es apropiada para reflejar un gran número de casos y es una forma efectiva de proveer niveles de abstracción.

El browser hipermedial esta diseñado para satisfacer este requerimiento, dado que permite navegar entre documentos de distinto nivel de abstracción utilizando el árbol de contenedores como guía. Uno de los objetivos mas importantes al incluir una herramienta de este tipo fue justamente el de ayudar al desarrollador en este sentido.

Cubrir el espectro completo de las actividades del desarrollo de sistemas de información.

El concepto de documento es lo suficientemente amplio para dar soporte desde las primeras etapas de análisis de requerimientos hasta la implementación y el mantenimiento. Un documento puede contener Especificaciones, diagramas , casos de test, grabaciones de entrevistas o código fuente, etc.

Otros requerimientos

Existen algunos requerimientos que fueron analizado y excluido de los alcances, al menos en este primera etapa. Estos requerimientos tienen algunas características comunes, que consideramos motivo suficiente para nuestra decisión.

- Son requerimientos comunes a varias áreas de la informática, no exclusivos de herramientas CASE.
- Tienen que ver con temas inmaduros que están actualmente siendo investigados
- Son de una complejidad importante.

- No son críticos para lograr nuestros objetivos.

Los requerimientos excluidos son:

Dar soporte a la actividad de varios usuarios y brindar a cada uno una visión de los desarrollos acorde con el rol que desempeña dentro del equipo.

Manejar versiones.

Soporte de varias tareas concurrentes.

Generación automática de código.

Funcionalidad del prototipo de HCase (HBrowser)

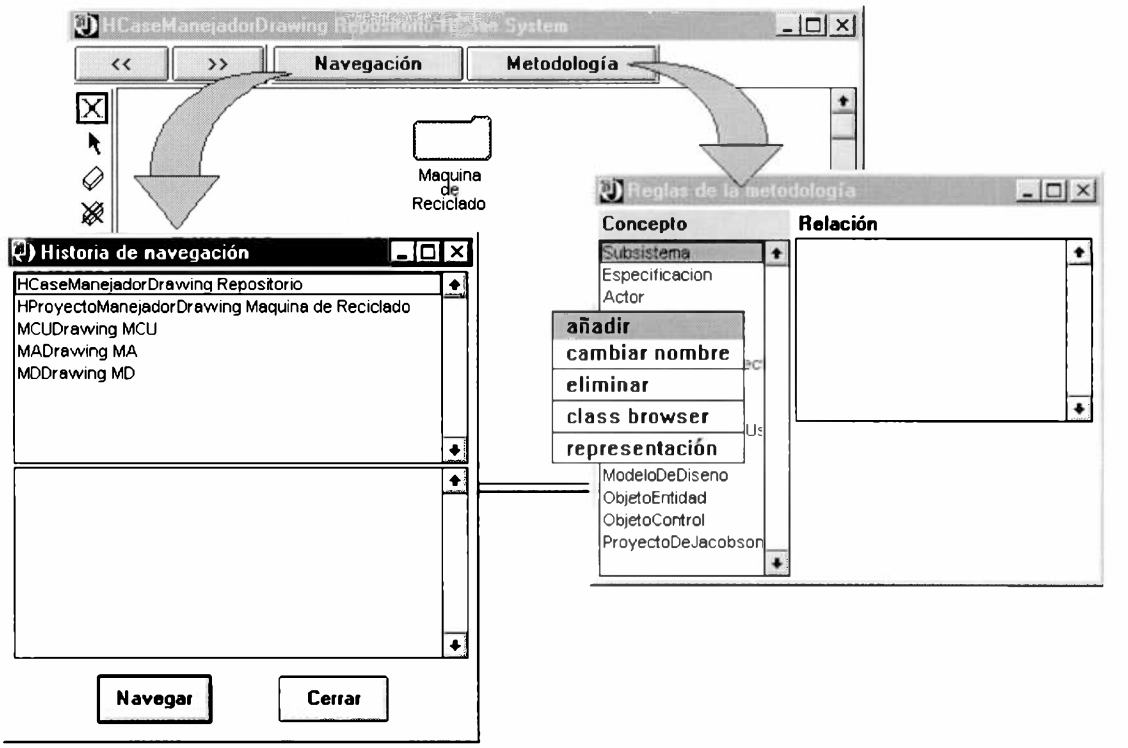
Introducción

La herramienta central que se desarrolló como interfaz de HCase es un browser hipermedial (HBrowser) que permite crear y realizar las actividades necesarias sobre el desarrollo del proyecto

HBrowser presenta facilidades para realizar dos actividades básicas sobre los documentos del repositorio, estas son:

- Exploración
- Edición de documentos

La exploración se realiza por medio de un browser hipermedial que permite navegar, entre documentos, utilizando las áreas sensibles que estos presentan. Como complemento cada tipo de nodo o documento cuenta con herramientas de edición apropiadas.



HBrowser tiene dos herramientas de entorno complementarias que permiten al espectador realizar las siguientes actividades:

- Consultar la pila de nodos visitados
- Personalizar la metodología

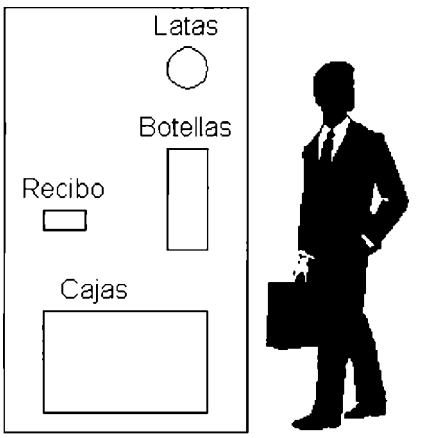
La primera apunta a solucionar el problema de desorientación del espectador en el hiperespacio, presentado una lista con los nodos visitados y permitiendo una navegación directa a cualquiera de ellos, la segunda, permite al espectador definir, adaptar y extender las reglas metodológicas soportadas por HCase.

Funcionalidad de HCase

Cabe la aclaración que la funcionalidad incluida en el prototipo de HCase y en particular en HBrowser es solo una parte de la que se previó en este trabajo. El motivo por el cual no se ha incluido toda la funcionalidad es el gran esfuerzo en horas de programación que se requeriría. Se ha incluido en el prototipo suficiente para mostrar la potencia de HCase y la forma en que se alcanzaron los objetivos de extensibilidad y adaptabilidad.

La máquina recicladora de Jacobson

Para la mejor descripción de la funcionalidad de HBrowser tomaremos un ejemplo simple y lo desarrollaremos utilizando la herramienta. Consideramos que el mejor ejemplo a seguir es el que sigue Jacobson en su libro "Object-Oriented Software Engineering. A Use Case driven Approach", dado que gráfica los conceptos más importantes de su metodología.



El sistema ejemplo controla una máquina de reciclado para botellas retornables, latas y cajas. El sistema puede ser utilizado por varios clientes y cada cliente puede retornar todos los tipos de ítems en cada ocasión. Además la máquina puede distinguir que clase de botella o lata esta reciclando.

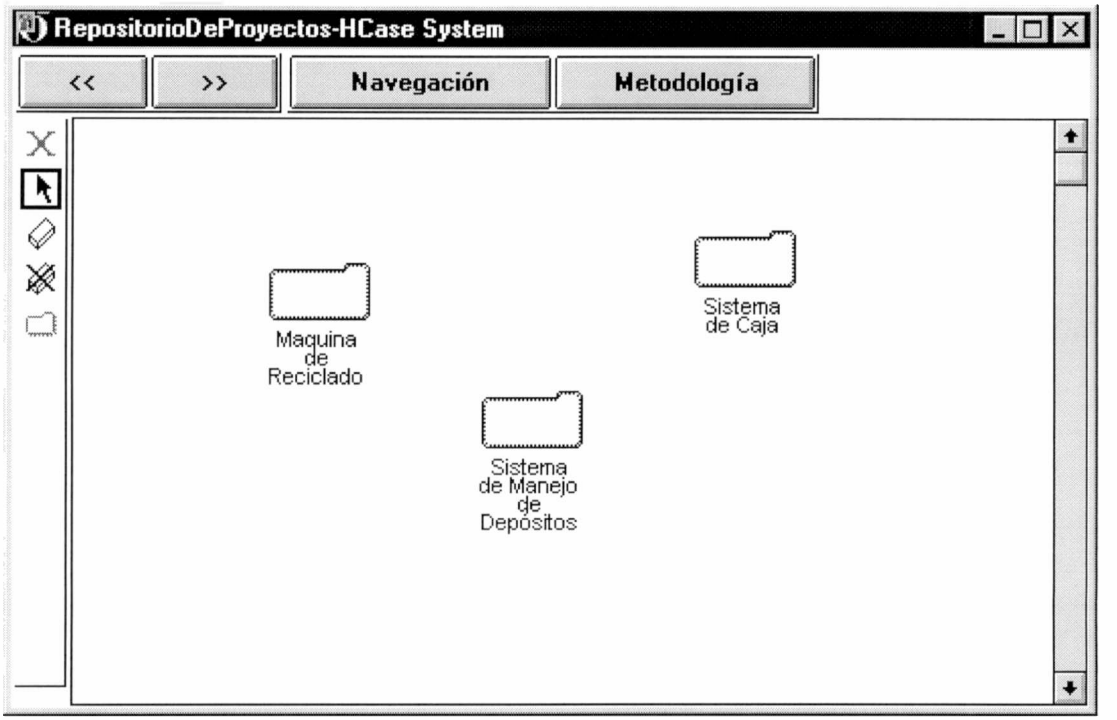
La máquina también emite un recibo para el cliente con un detalle de lo reciclado y puede llamar a un operador cuando un ítem esta atascado, además emite un "Cierre de caja" con un detalle de lo recibido en el día.

Documento del Repositorio de proyectos

HBrowser abre inicialmente en un nodo especial llamado "Repositorio de proyectos", este nodo es creado con HCase y no puede eliminarse, asegurando la existencia de un hiperespacio (un nodo) para explorar.

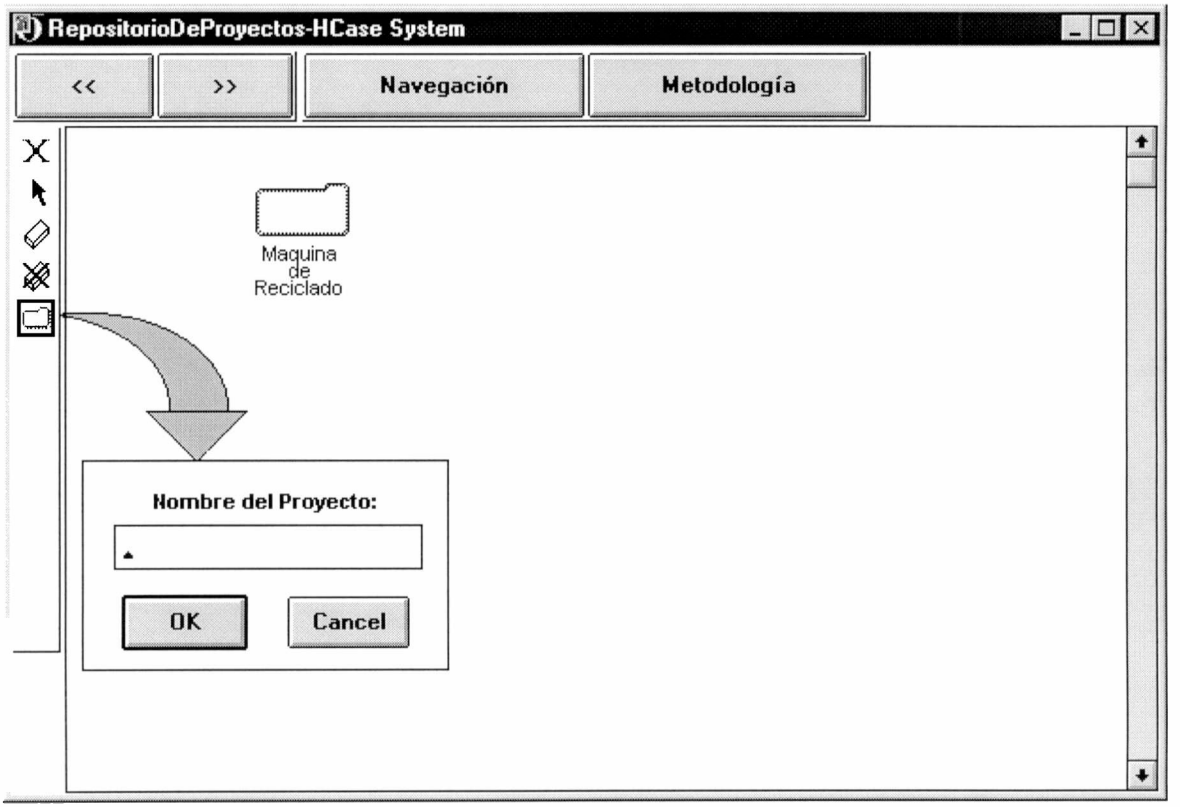
El repositorio de proyectos es único para todo HCase y contiene TODOS los proyectos que, en él, se desarrollan.

El documento del repositorio muestra una carpeta por cada proyecto de HCase y la funcionalidad del editor tiene facilidades para la creación y eliminación de proyectos.



Herramienta de Creación y Creación del primer proyecto

La herramienta de creación  permite crear carpetas de proyecto.



Dado que el repositorio de proyectos es, inicialmente, el único nodo del hiperespacio comenzaremos utilizando las actividades de edición para crear nuevos

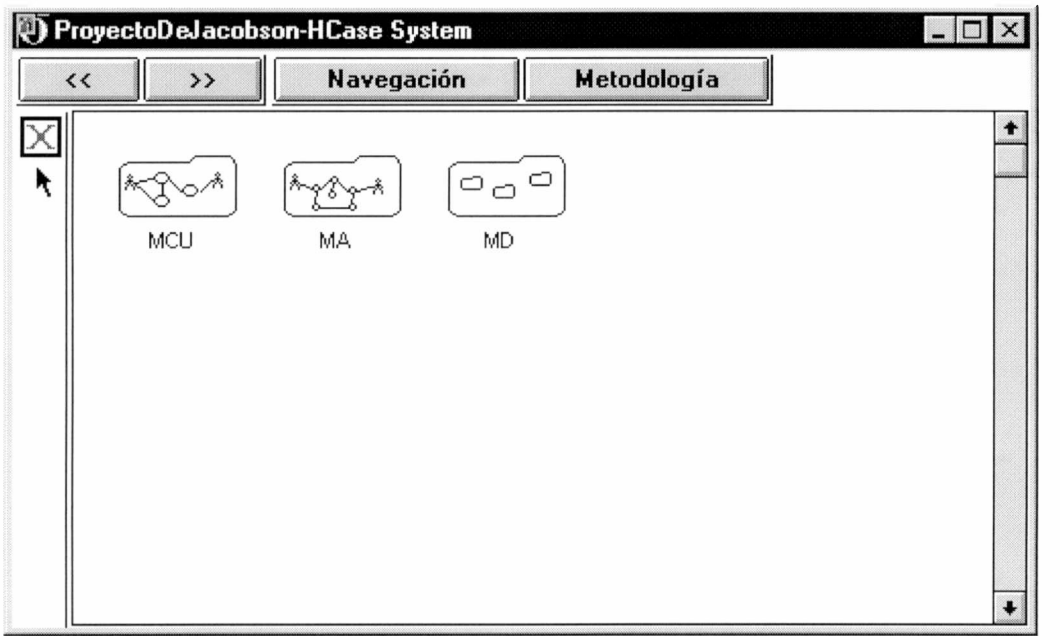
documentos y así iniciar la exploración. La primer actividad de edición es la creación del proyecto "Maquina de reciclado", para lo cual utilizamos la herramienta correspondiente y creamos la "Carpeta" del proyecto haciendo un click en la herramienta, un click en el documento y poniendo el nombre del proyecto.

Una vez creado el primer proyecto podemos comenzar la creación, exploración y edición de los nuevos documentos.

Herramienta de Navegación de contenidos

Junto con la carpeta del proyecto "Maquina recicladora" se crean tres carpetas con los "Modelos" del proyecto:

- Modelo de casos de uso
- Modelo de análisis
- Modelo de diseño



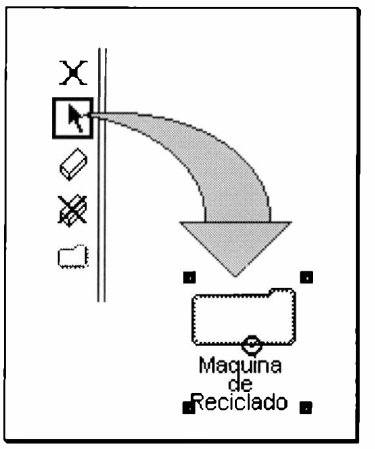
La herramienta de navegación de contenidos permite seleccionar un proyecto y navegar hacia el documento del proyecto, el cual tiene una carpeta para cada modelo.

Esta es una de las herramientas mas importantes, ya que permite visualizar mas detalles sobre el documento con el que se esta trabajando. Esta herramienta permite navegar hacia el interior de cualquier proyecto introduciéndose en los detalles de cada modelo y cambiando de nivel de abstracción.

Herramienta de selección

La herramienta de selección permite seleccionar una carpeta de proyecto o cualquier otra figura que aparezca en un documento con el objetivo de moverla o realizar otra acción sobre ella.

Al seleccionar una figura se hacen visibles los "handles", estos permiten realizar una variedad de acciones sobre las figuras. Aunque en el caso de las carpetas de proyecto no se requirió ninguna acción, en otros documentos se verán su funcionalidad con mas detalle.



Herramientas de borrado

Presentación y concepto de figuras

Antes de continuar con la explicación de las facilidades de edición es necesario introducir una decisión de diseño, tomada con respecto a las figuras que aparecen en los documentos, que afecta a la funcionalidad de HCase.

Cuando analizamos las metodologías y los documentos que estas generaban encontramos dos patrones que se repetían a menudo:

- Elementos que se repiten en varios documentos, por ejemplo clases o actores.
- Objetos que son presentaciones que muestran aspectos distintos del mismo concepto (Tarjeta de clase, Interfaz de clase, Estructura interna de una clase)

Basándonos en estas características decidimos que cada figura y cada documento de HCase posea dos componentes, una presentación y un concepto. Utilizando esta estructura modelamos los casos anteriores.

Si llamamos a las figuras y a los documentos elementos perceptibles, la presentación intenta manejar todas las responsabilidades, del elemento perceptible, que tienen que ver con su presentación, mientras que el concepto intenta capturar las responsabilidades y atributos que el elemento mantiene independientemente de la forma en que este se perciba.

El objetivo de esta decisión es que un objeto de diseño sea compartido por varios documentos. de esta forma la consistencia es mantenida en forma natural (El caso mas importante es el de una clase compartida por varias aplicaciones o proyectos).

Mas adelante profundizaremos sobre este tema.

Herramienta de borrado "Soft"

La herramienta de borrado "Soft" permite borrar una carpeta sin borrar "realmente" el proyecto, mas tarde se puede crear una nueva carpeta que permita visualizar nuevamente el proyecto.

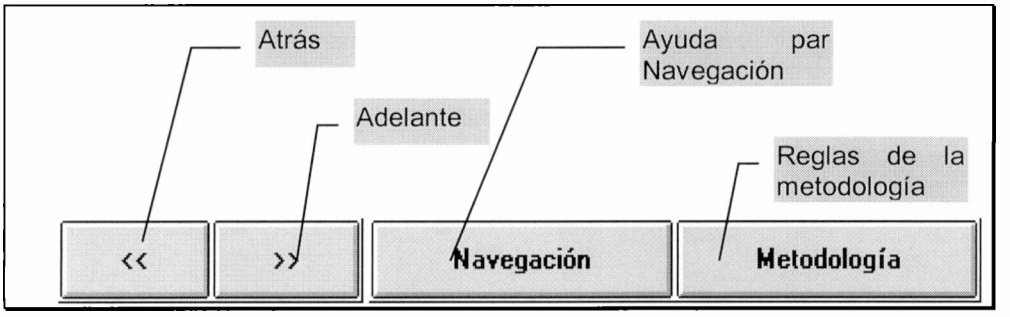
Borra solo la presentación de un elemento de manera que no aparece mas en ese documento. Esta herramienta deja intacto el concepto, de manera de poder utilizarlo en otro documento.

Herramienta de borrado "Hard"

Al utilizar la herramienta de borrado "Hard" se borra "definitivamente" el proyecto de forma tal que no puede ser recuperado.

Borra la presentación y el concepto. Esto implica la eliminación de todas las presentaciones correspondientes al concepto. Y la eliminación total del objeto dentro de los alcances de HCase .

Botones de Navegación por la pila y personalización de HCase



Pila de nodos visitados

HBrowser guarda una pila con los nodos visitados. Cada vez que se navega a un nuevo nodo se incorpora al tope de la pila, manteniendo el orden cronológico. HBrowser puede mostrar la pila de nodos navegados permitiendo seleccionarlos y navegar a cualquiera de ellos. También permite movimientos secuenciales, moviéndose hacia adelante o atrás en el orden cronológico en que los nodos fueron visitados.

Botones << y >>. (Atrás y Adelante)

La pila posee un cursor que indica el nodo actualmente visitado, los botones Adelante y Atrás permiten mover el cursor por la pila y navegar hacia adelante o hacia atrás en el orden cronológico en que los nodos fueron visitados.

Para probarla podemos navegar, utilizando la herramienta de contenidos, hacia el modelo de Casos de uso y jugar yendo y volviendo al nodo de proyecto.

Botón Navegación

Esta herramienta abre una ventana que muestra la pila de nodos en el orden en que fueron visitados, el usuario puede seleccionar cualquiera y navegar hacia el.

La forma de probarlo es abriéndola y navegando a cualquier nodo que se desee.

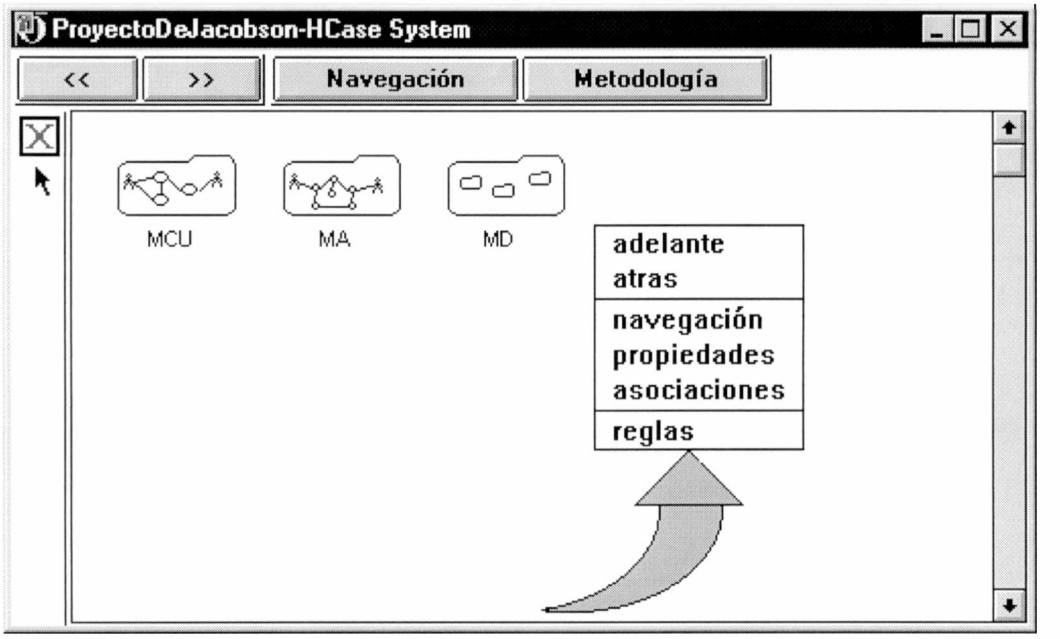
Botón Metodología

Este botón permite abrir una herramienta cuyo objetivo es mostrar y permitir modificar las reglas que permiten asociar a los diferentes documentos y objetos creados durante el desarrollo.

Las reglas de asociación son fundamentales para el éxito de la herramienta ya que permiten controlar la consistencia de los documentos y controlan los criterios de navegación entre documentos. Luego veremos en mas profundidad la utilización de estas reglas y como un conjunto demasiado chico o grande de reglas puede perjudicar a la aplicación de la metodología.

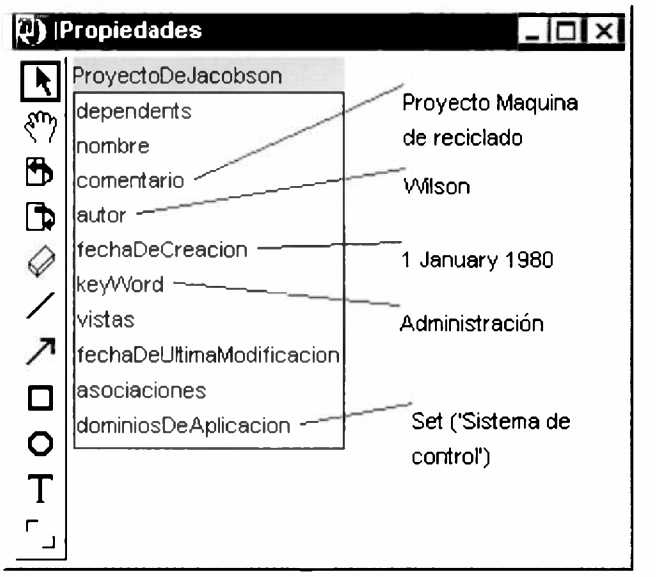
Mas adelante ampliaremos la descripción de esta herramienta.

Menú del documento



Menú de Propiedades

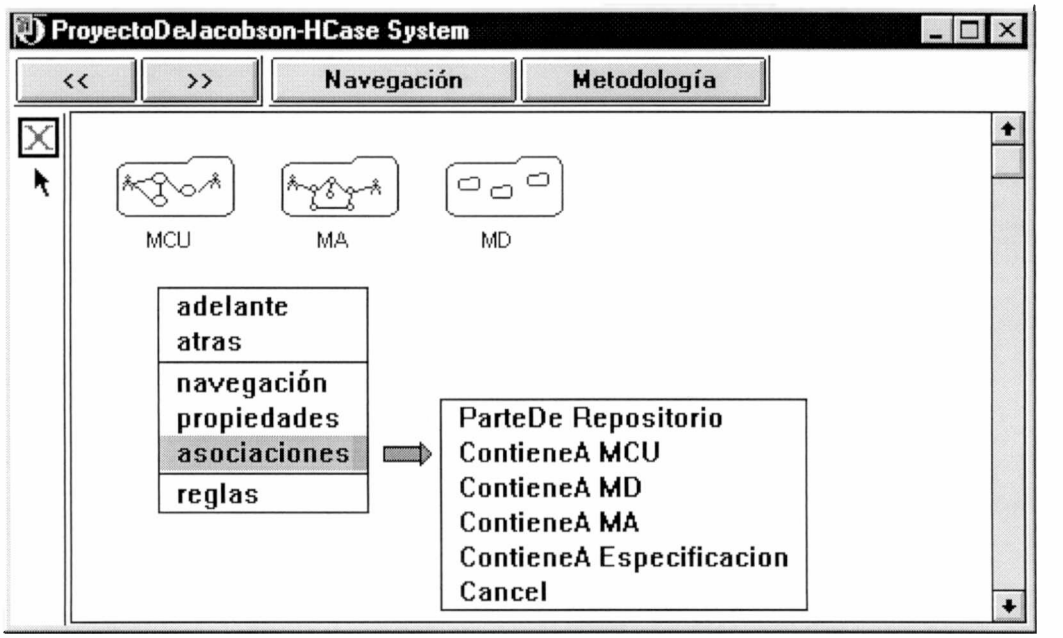
Cada nodo que visitamos posee un conjunto de propiedades que pueden ser de utilidad para el desarrollador. Algunas de estas propiedades son datos de gestión que se calculan automáticamente, otros pueden ser manejados por el usuario.



Autor, fecha de creación, cantidad de veces que fue visitado, comentarios, cantidad de objetos contenidos y otros datos pueden ser útiles para la utilización de métricas y como criterios para búsquedas de objetos reusables.

Cuando navegamos a las propiedades de un nodo, el nodo destino es uno en el que se muestran algunos datos de utilidad sobre el nodo que estabamos viendo.

Menú de Asociaciones



Que un documento este contenido en otro o que me permita ver las propiedades de un nodo no son los únicos criterios que hacen deseable navegar de un nodo a otro.

Este es el caso típico en que estamos parados en una tarjeta de clase y queremos ver información de otra que colabora con ella, en este caso podemos aprovechar la existencia de la asociación "Colabora con" y navegar a la tarjeta de clase deseada. Para

el caso del ejemplo Encontraremos que el nodo Repositorio De Proyectos tendrá una asociación "Compuesto por" con cada uno de los proyectos.

El menú de asociaciones permite seleccionar cualquier asociación existente entre el documento corriente y cualquier otro documento del repositorio y utilizarla como link para navegar hacia el documento asociado.

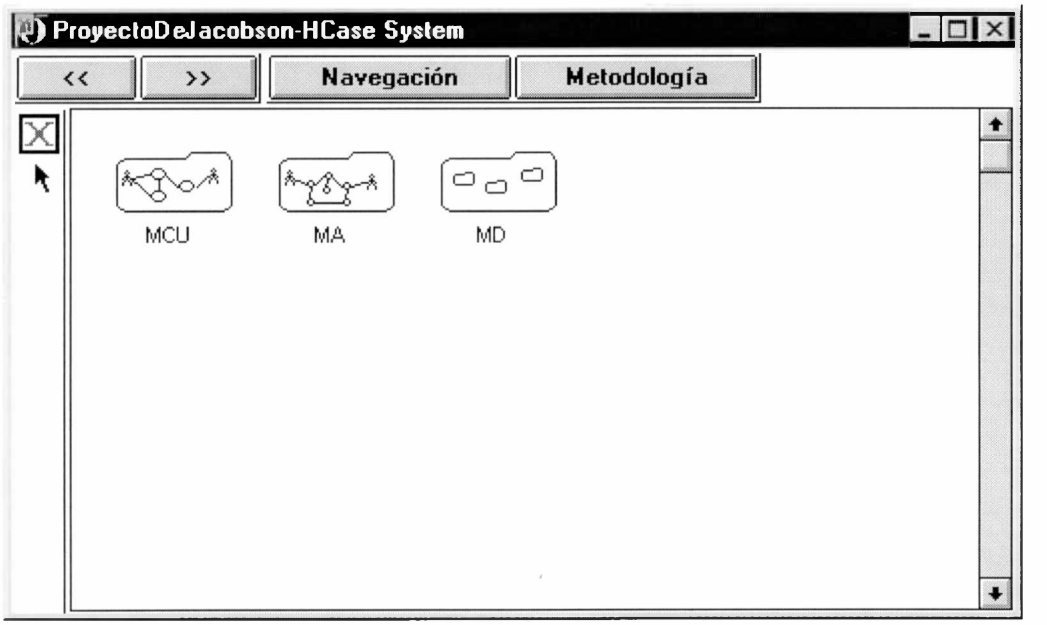
Documento del proyecto

La barra superior de herramientas pertenece al browser y no cambia entre documentos, por lo tanto asumiremos que siempre esta presente independientemente del documento (nodo) en el que estemos. Una asumción similar se puede hacer con respecto al menú del documento.

Las diferencias entre los documentos surgen fundamentalmente por las figuras que se permiten incluir en cada uno y la funcionalidad que estas presentan.

Aunque existen otras razones, la barra de herramientas lateral presenta variantes que se deben principalmente a las distintas herramientas de creación, efecto directamente relacionado con la cantidad y tipo de figuras permitidas.

Por todas estas razones describiremos para cada documento, fundamentalmente, las figuras y el comportamiento que estas presentan en cada caso. Y asentaremos puntualmente cualquier variación en otros aspectos.

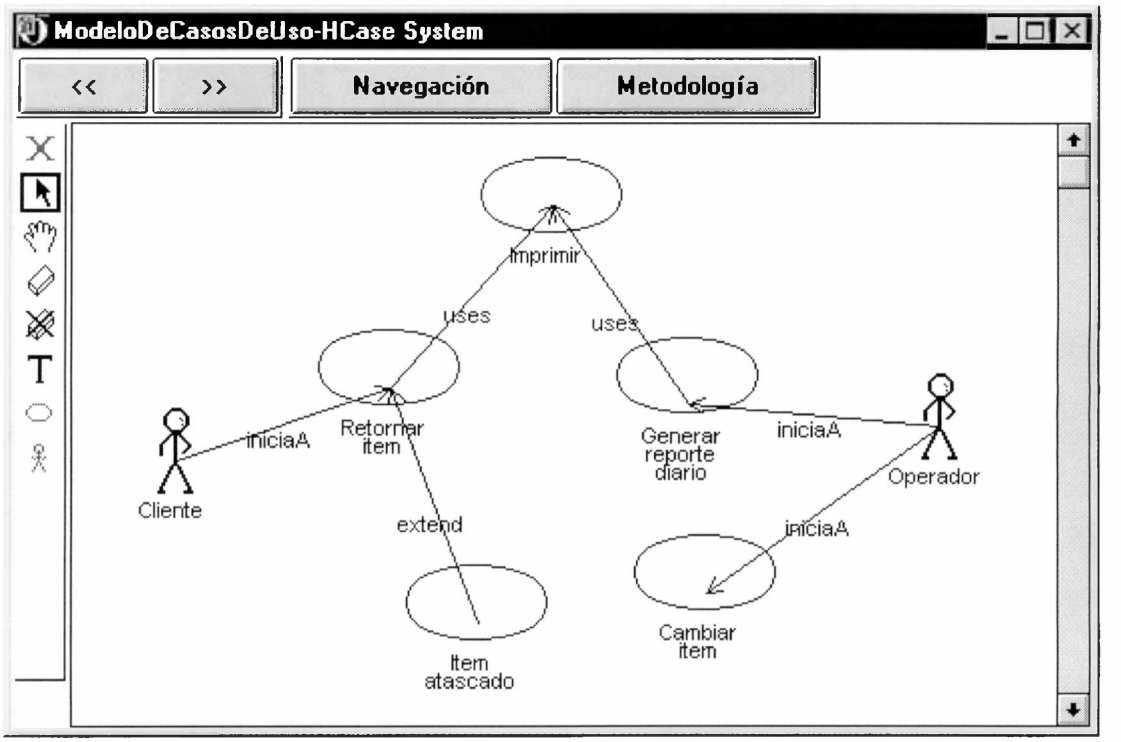


El documento de proyecto tiene el objetivo de mostrar una visión a muy alto nivel de abstracción del proyecto, la idea es mostrar la forma en que está organizado el proyecto y permitir la mejor ubicación del desarrollador dentro del repositorio. Las únicas herramientas incluidas son la de selección y contenidos dado que las figuras contenidas se crean junto con el proyecto y solo pueden ser movidas de lugar.

En los tres modelos puede adicionarse textos aclaratorios y comentarios.

Documento del modelo de requerimientos

El documento del modelo de requerimientos esta diseñado para construir modelos de casos de uso. Las figuras representativas de este modelo son **Actor** y **Use case**.



La herramienta "scroll"  permite mover todo el gráfico.

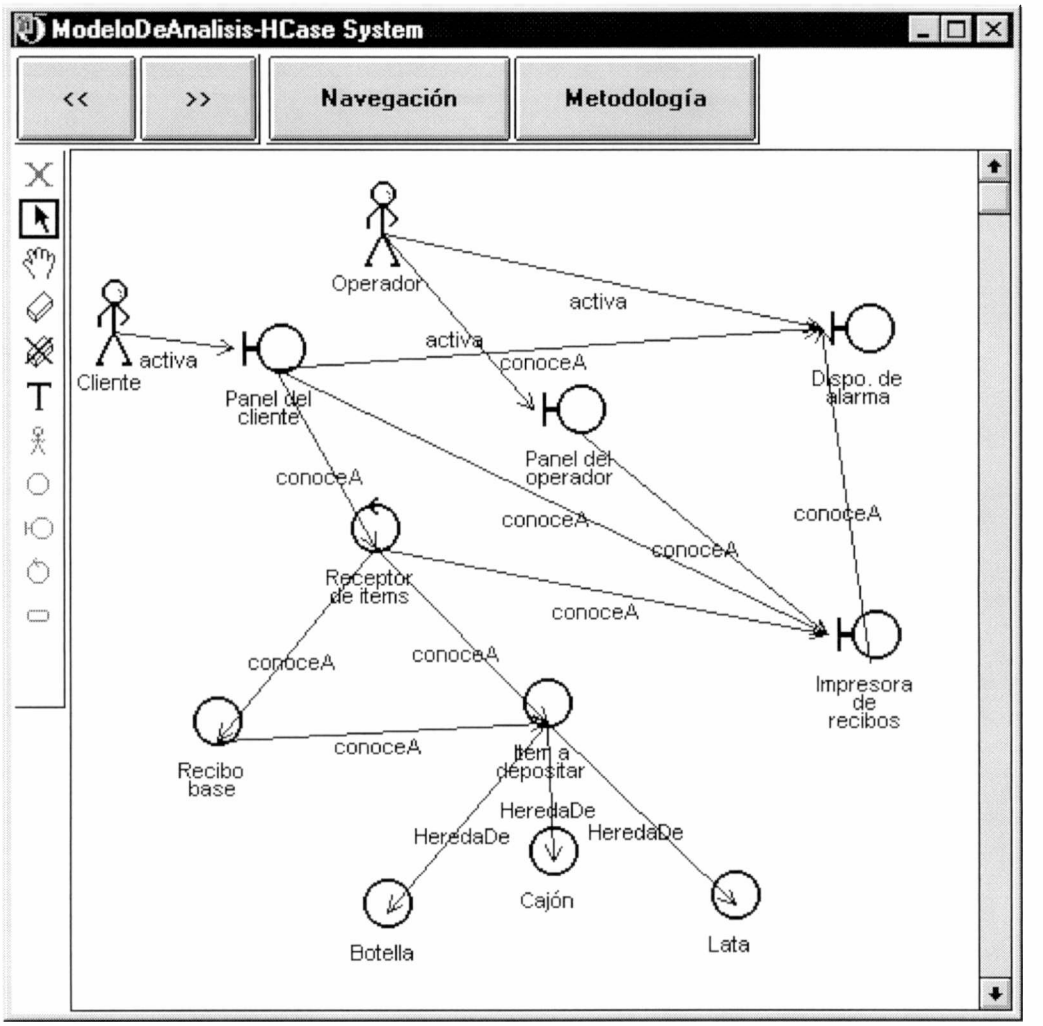
Con la herramienta de selección  se permite mover figuras, además de permite establecer relaciones entre las figuras según lo indica la metodología. El arco "Acompañan" a las figuras cuando estas son movidas.

Crear un Use Case 

Crear un Actor 

Documento del modelo de análisis

La funcionalidad de este modelo es análoga a la del modelo de requerimiento, pero cambian las figuras y las relaciones que pueden existir entre ellas. En este modelo se permiten crear figuras del modelo de análisis, **objeto interfaz**, **objeto control** y **objeto entidad**, así como relacionarlos según lo indica la metodología.



Crear Actor 

Crear Objeto entidad 

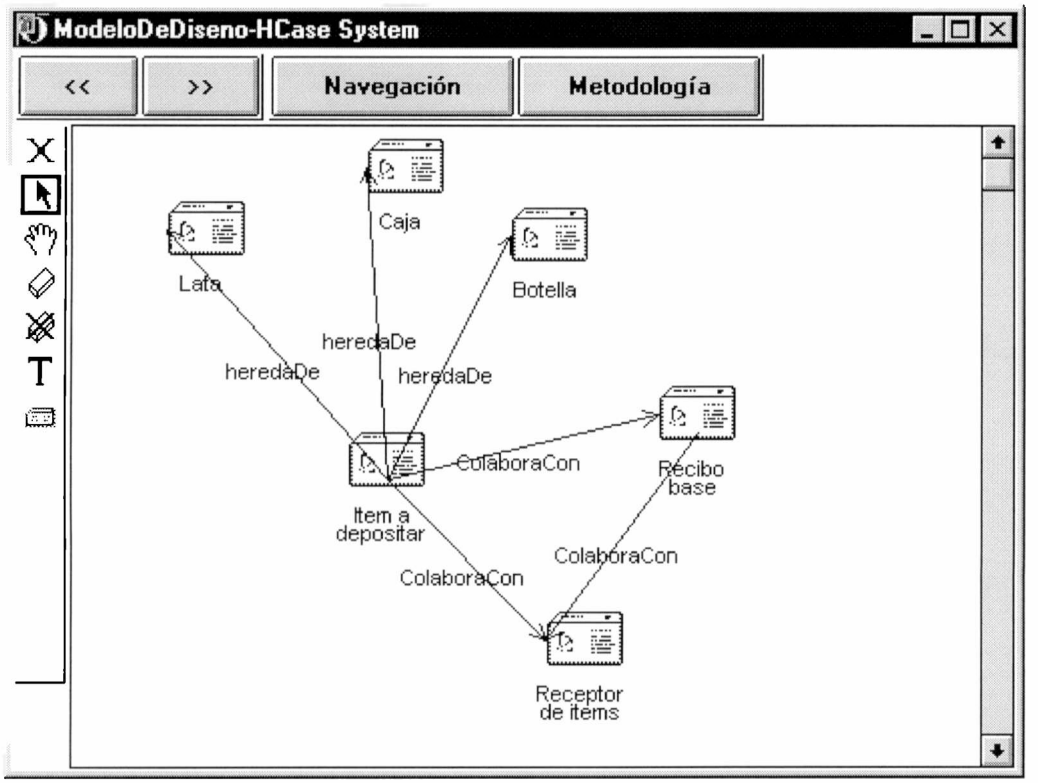
Crear Objeto interfaz 

Crear Objeto control 

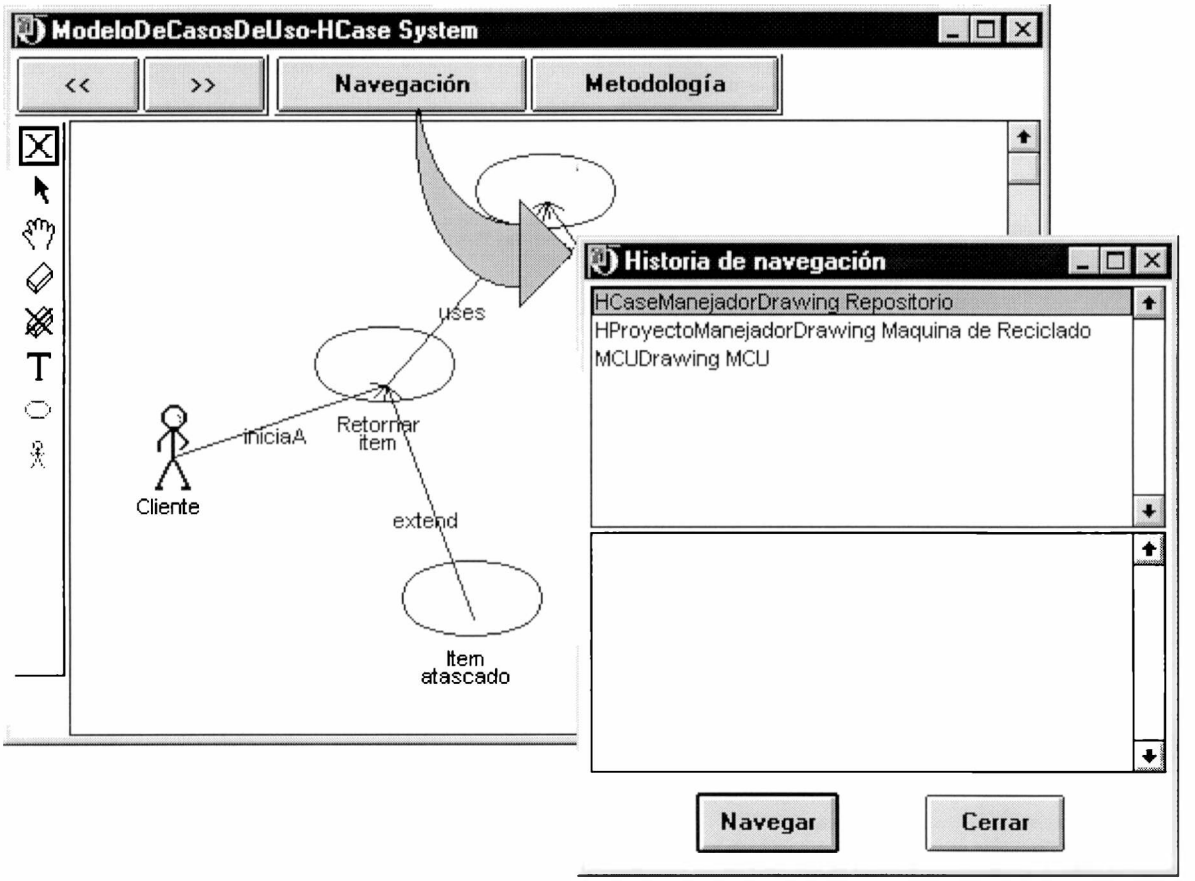
Documento del modelo de diseño

Este modelo permite modelar el conjunto de tarjetas de clase y las colaboraciones entre ellas, la funcionalidad es análoga a los documentos de análisis y diseño

Crear Tarjeta de clase 



Herramienta de navegación



La herramienta de navegación es una herramienta muy sencilla que permite visualizar una lista con los nodos visitados en forma cronológica, de tal forma que el usuario pueda seleccionar un nodo y navegar hacia él.

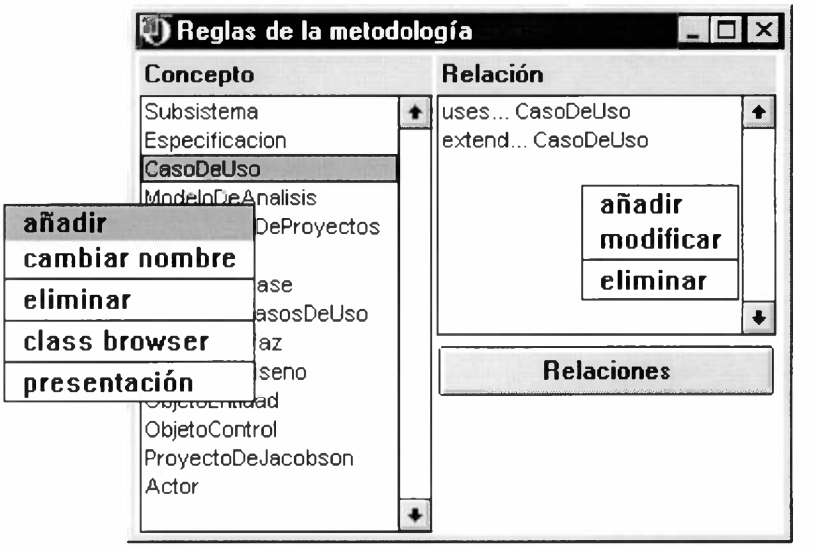
Herramienta de reglas de la metodología

Anteriormente dijimos que cada documento y cada figura se componía de 2 elementos, la presentación y el concepto. La forma en que la metodología define como se relacionan las figuras y documentos esta modelada a nivel de conceptos. La herramienta de la metodología permite personalizar el comportamiento de los distintos elementos de la metodología tanto a nivel de concepto como de presentación.

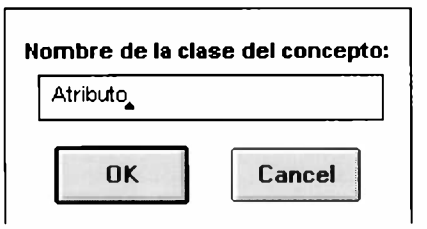
A nivel de concepto, la herramienta de la metodología, permite agregar nuevos conceptos (Para crear nuevos documentos y figuras), modificar algunas propiedades y eliminar conceptos. También se permite ver las presentaciones asociadas al concepto y editar la clase que genera, para modificar su comportamiento.

La herramienta de reglas de la metodología permite, también, armar 3-uplas que especifican la forma en que dos conceptos se pueden relacionar, por ejemplo "Actor - IniciaA - Caso De Uso", o "Clase - Colabora con - Clase"

Estas reglas se incorporan a la base de conocimiento de la metodología y son tomadas por los diferentes editores para establecer cuales son las relaciones permitidas entre figuras.

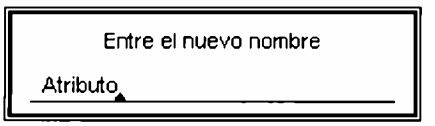


Añadir concepto



Para añadir un concepto solo hace falta seleccionar el ítem de menú correspondiente y completar el nombre del nuevo concepto. El único requisito solicitado es que el nombre del nuevo concepto no sea igual al nombre de un concepto existente.

Renombrar concepto



Renombrar el concepto es similar a la creación. Es importante que lo que estamos nombrando es la clase de conceptos, no un concepto en particular. Si modificáramos el nombre del concepto "Actor" estaríamos modificando todos los actores de la herramienta y no un actor en particular.

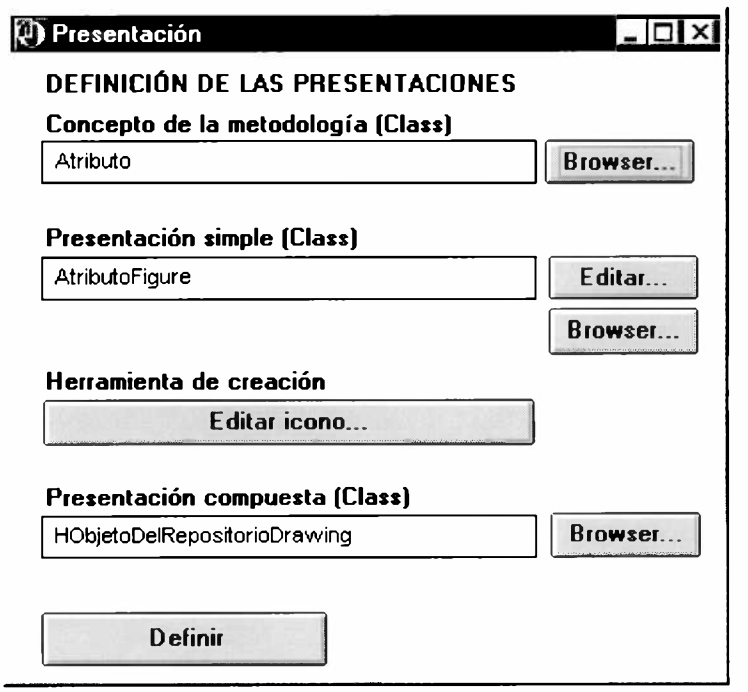
Eliminar un concepto

Esta funcionalidad no ha sido completamente desarrollada ya que no se decidió que hacer con las instancias del concepto y con las presentaciones asociadas, si las hubiera.

Abrir el Class Browser

Cada tipo de concepto tiene asociada una clase en la jerarquía de conceptos, la herramienta de la metodología permite utilizar el Class Browser para mostrar la definición de la clase correspondiente al concepto. Dentro de la funcionalidad del Class Browser el usuario puede personalizar el comportamiento del concepto.

Editar la presentación



La herramienta de presentación permite, dado un concepto:

Tener acceso a la clase del concepto

Definir una clase de figura que represente a ese concepto

Construir la clase de herramienta que permita a un documento crear este tipo de figura

Definir una clase de documento que muestre algunos aspectos del concepto

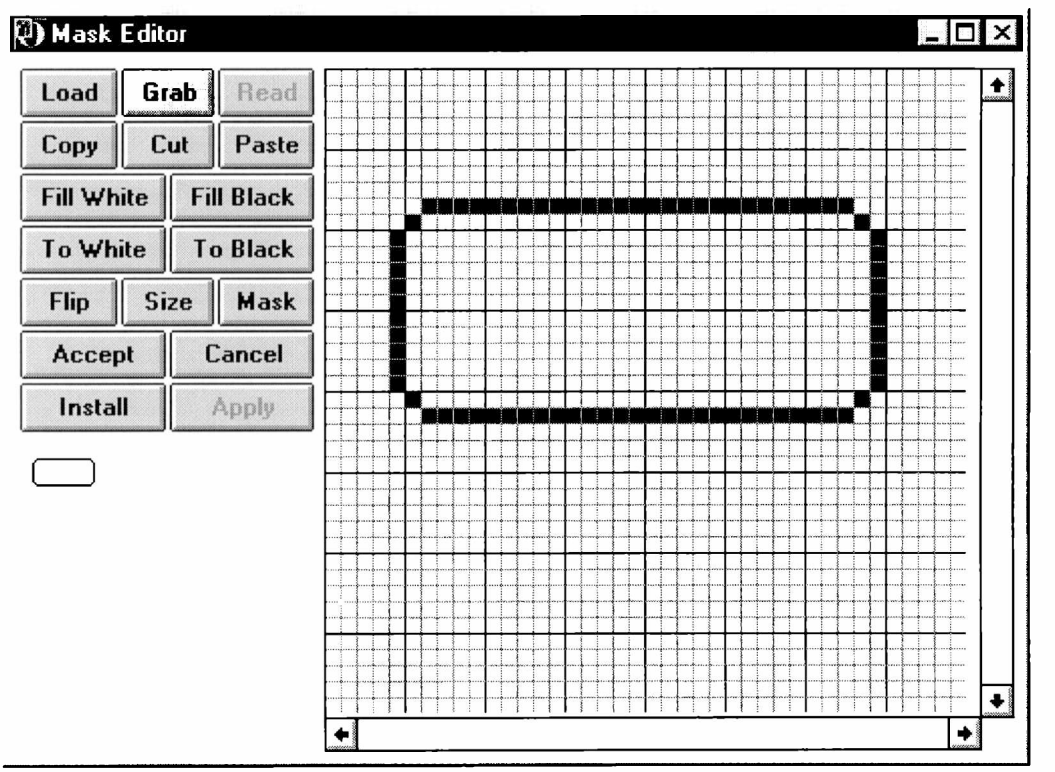
Acceder al Class Borwser para personalizar el comportamiento del documento.

Definir presentación simple o figura

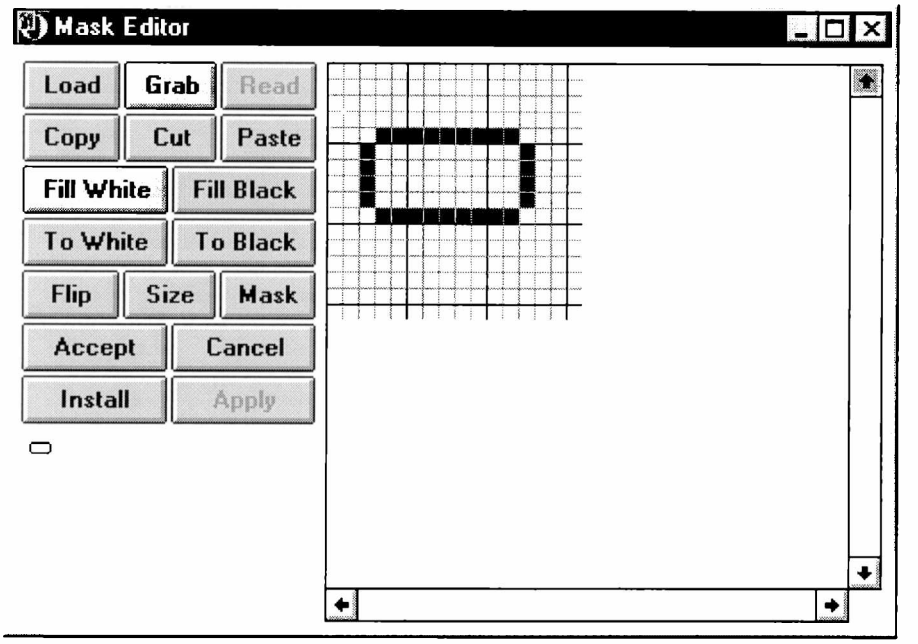
En la presentación simple se define el nombre de la clase con el que se implementa la presentación. Definir una nueva clase de figuras requiere varias tareas, la primera es definir el concepto, las otras son:

- Editar la figura principal
- Instalar la clase de la figura
- Editar el ícono de la herramienta de creación
- Instalar la clase de la herramienta

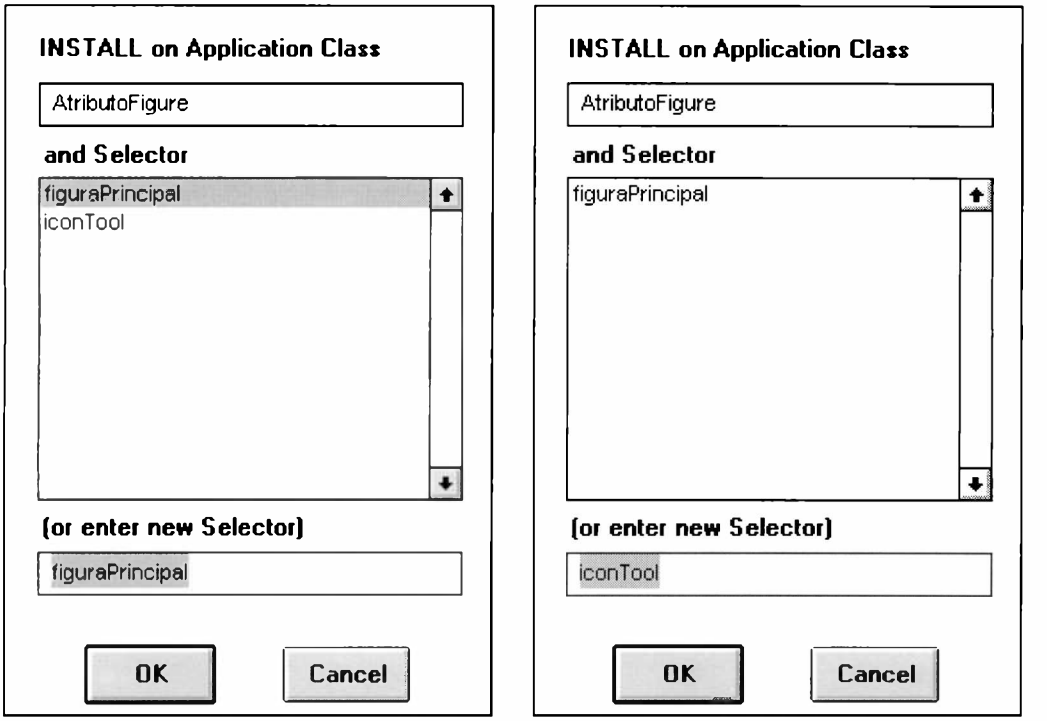
Editar figura principal y el ícono de la herramienta



Para editar la figura principal y el ícono de la herramienta de creación se utiliza un editor gráfico que provee las facilidades necesarias para esta tarea.



Instalar la Presentación y la herramienta



Luego de editar la figura principal y el ícono deben se instaladas las clases correspondientes.

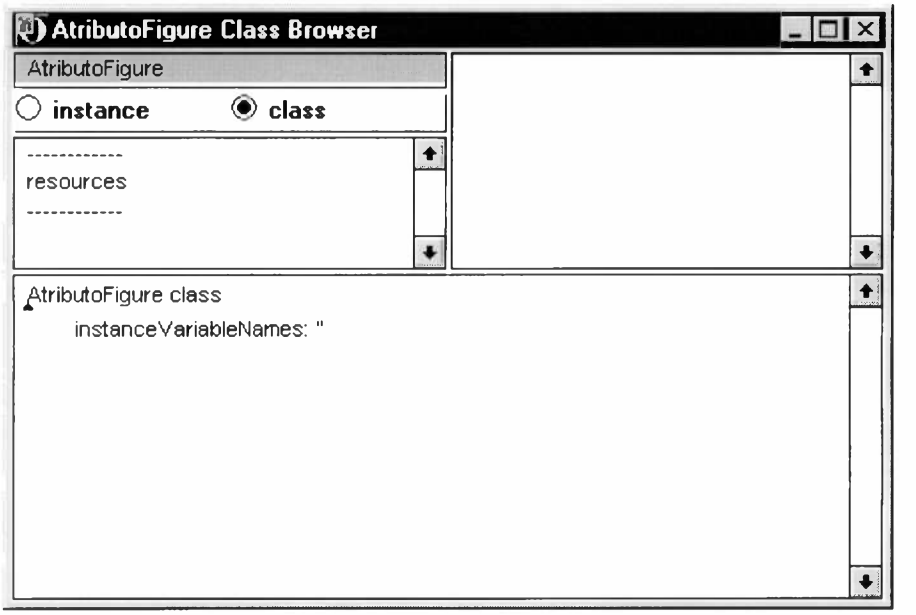
Definir presentación compuesta o Diagrama

En la presentación compuesta o diagrama se define el nombre de la clase con el que se implementa la presentación.

Hasta el momento no se han creado facilidades para que el usuario pueda definir el comportamiento sin necesidad de trabajar directamente sobre la definición de la clase,

pero esto es perfectamente posible y se tendrá en cuenta en la sección de extensiones posibles.

Para cada uno de los elementos definidos se permite acceder al Class Browser para especializar determinados mensajes.



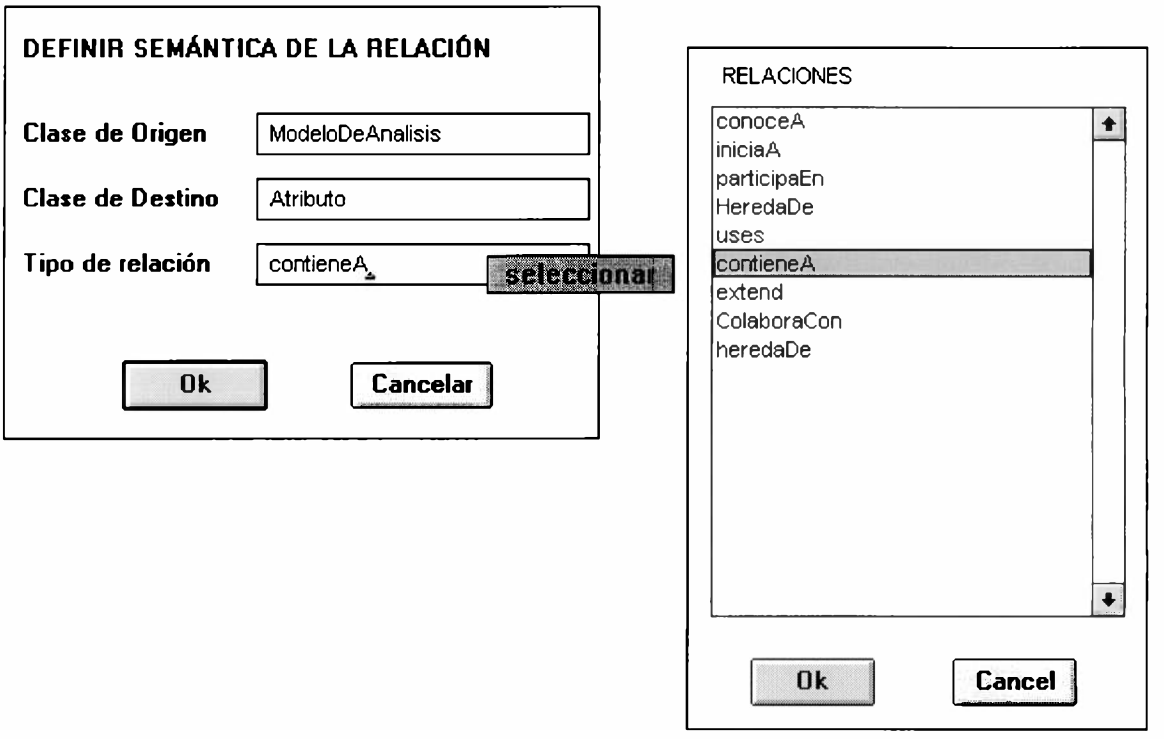
Añadir relación al concepto

Dado un tipo de concepto es deseable definir la forma en que se asocia con otros tipos de conceptos, en las figuras vemos las herramientas que permite definir esta 3-uplas:

- Se indica el concepto origen

- Se selecciona el tipo de concepto destino de una lista de conceptos posibles.
- Se selecciona el tipo de asociación que puede existir entre ellos.

Estas 3_uplas También se puede eliminar y modificar.



Extensiones previstas a la funcionalidad HCase

Si bien la herramienta principal de HCase es HBrowser como parte del proyecto analizamos y especificamos otras herramientas y funciones que serían apropiadas para que el CASE cumpla con los requerimientos previstos.

Queremos aclarar nuevamente que estas herramientas no se encuentran en el prototipo que construimos pero su inclusión sería perfectamente posible con el esfuerzo de programación correspondiente.

Navegar por links definidos por el espectador

Si bien HCase provee un conjunto de links predefinido de acuerdo a la metodología, seria de utilidad contar con links cuya semántica sea determinada por el espectador. Este tipo de links existiría al solo efecto de agilizar la navegación. No tendría semántica dentro de la metodología, pero permitiría reflejar el pensamiento poco estructurado del usuario.

Navegar documentos externos - Editor del contenido

Este tipo de navegación permitiría integrar, HCase con otras herramientas del ambiente. Por ejemplo si estamos viendo un documento de una clase, seria deseable poder abrir el browser de clases para editarla, o si estamos viendo una descripción de interfaz abrir el "Canvas" (El "Canvas" es una herramienta de Objects Works para desarrollar ventanas en forma visual).

Anotate

Una característica de los browsers hipermediales es que permiten agregar anotaciones a los documentos que se estén explorando, seria deseable que con un click en la figura que representa la anotación abrir inmediatamente un editor para ella, que la mostrara y permitirá editarla.

Bookmark

Cuando un nodo es de interés y se desea marcarlo para poder retornar rápidamente, se puede marcar, al marcarlo el nodo se agregara a una lista de nodos marcados a la que se puede acceder y de la cual se puede seleccionar un nodo al cual navegar. Esta funcionalidad aportaría a la ubicación del usuario.

Links desde el texto (HotWords)

Cuando se navega por descripciones textuales es muy frecuente que algunas palabras sean significativas y que por tal permita navegar a otro nodo donde se encuentre mas información al respecto. En un desarrollo de software es muy frecuente encontrar descripciones textuales y sería útil permitir navegar a otro nodo utilizando la palabra como área sensible, este tipo de link se llama HotWord y debería ser soportado por el browser hipermedial.

Generación automática de código

Si bien la primer versión de HCase llega hasta la etapa de diseño, sería posible utilizar las facilidades que provee el ambiente e incorporar la etapa de construcción a HCase. La primer idea que surge es la de utilizar el conjunto de nombres generados en las primeras etapas como base para la construcción, los nombres de los objetos pueden derivarse en nombres de clases, las colaboraciones en responsabilidades, etc.

Esta facilidad sería extremadamente potente ya que evitaría la duplicación de mucho trabajo, evitaría inconsistencias entre la documentación y el desarrollo y facilitaría la documentación.

Dar soporte a nuevos documentos

Existe varios documentos previstos por la metodología que no fueron incluidos. Sería bueno agregar editores y otras facilidades para ellos. Además se previó la construcción de una herramienta que permita definir estos documentos dentro del ambiente. Los documentos son:

Especificación de requerimientos

Modelo de objetos del dominio del problema.

Diagrama de interacción

Diagramas de bloques

Diagramas de estados y transiciones

Diagramas de flujo

Modelo de test

Chequeo de reglas de consistencia

Existe infinidad de reglas de consistencia que deberían ser respetadas, Herencia múltiple, Actores que no participan en ningún Caso de Uso, Casos de uso que no fueron desarrollados a través de los modelos, etc. Sería útil hacer una herramienta que permita especificar y chequear estas reglas de consistencia.

Otros Browsers

Una herramienta que permita navegar por los distintos documentos es importantísima pero existe otras formas de acceso que también son importantes y poderosas a la hora de realizar desarrollos de software. Existen áreas como métricas, gestión de proyectos o reuso de componentes que requieren formas distintas de manipular la información, por esto es deseable la existencia de browsers apropiados que den soporte a estas actividades.

Dado que el desarrollo producido con HCase puede verse como una red hipermedial, es frecuente (y útil) utilizar mapas tipo grafo para facilitar la orientación del espectador, esta es otra posible extensión a HCase.

Otra visión que se puede tener de los desarrollos producidos por HCase es la de una base de datos con información de los desarrollos, es posible desarrollar una herramienta que permita hacer Querys para poder gestionar los proyectos.

Clasificación jerárquica facetada, que son browsers que muestran el conjunto de documentos clasificados de diferente forma y permitiendo seleccionar uno al cual navegar.

Debido a los múltiples puntos de vista con los que se puede ver al repositorio es posible extender la funcionalidad de HCase construyendo distintos browser según las necesidades del usuario.

Análisis del impacto de las modificaciones

Los distintos elementos que aparecen en los desarrollos están asociados entre sí, teniendo suficiente información de la forma en que se asocian podemos definir reglas que determinen el impacto que produce la modificación de un elemento en los elementos asociados.

Otra extensión importante podría ser la construcción de una herramienta que utilice estas reglas para analizar el impacto de los cambios producidos al modificar algún elemento de software.

Reuso de conceptos existentes

Una de las funcionalidades principales que pretendemos de HCase es la posibilidad de reusar los elementos desarrollados, una extensión que debe hacerse a HCase es la de permitir buscar y reutilizar elementos ya desarrollados.

Roles del usuario

La modalidad de trabajar en grupos es fundamental para lograr desarrollos de envergadura. Para lograr productos de alta calidad sería útil contar con capacidades de seguridad y diferentes presentaciones de acuerdo a los roles de los usuarios.

Otras metodologías

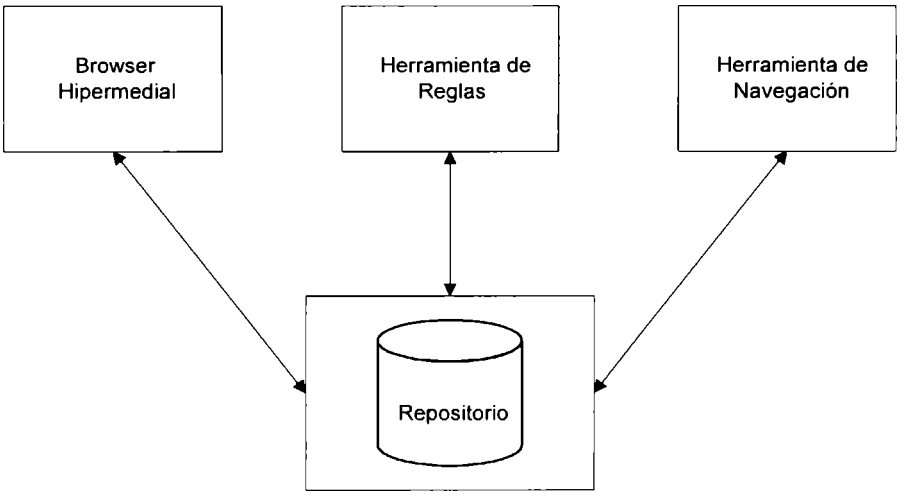
Adaptar el HCase a otra metodología significa fundamentalmente, agregar nuevos tipos de documentos, nuevos tipos de asociaciones, implementar las reglas de la metodología y adaptar las herramientas para trabajar con los nuevos elementos. Esta capacidad se logra como composición de las funcionalidades anteriores.

Arquitectura de HCase

HCase se compone de un repositorio común y un conjunto de herramientas integradas.

En el repositorio encontramos la información de los distintos desarrollos de software por ejemplo descripciones de procesos, descripciones de interfaces, especificación de clases, gráficos, configuraciones de software y librerías de documentos en general.

Sobre la estructura del repositorio se asienta un conjunto de herramientas. Las herramientas están diseñadas para trabajar en base a los objetos del repositorio, los crean, utilizan y actualizan su contenido. De esta manera se generan desarrollos nuevos y se mantienen los existentes.

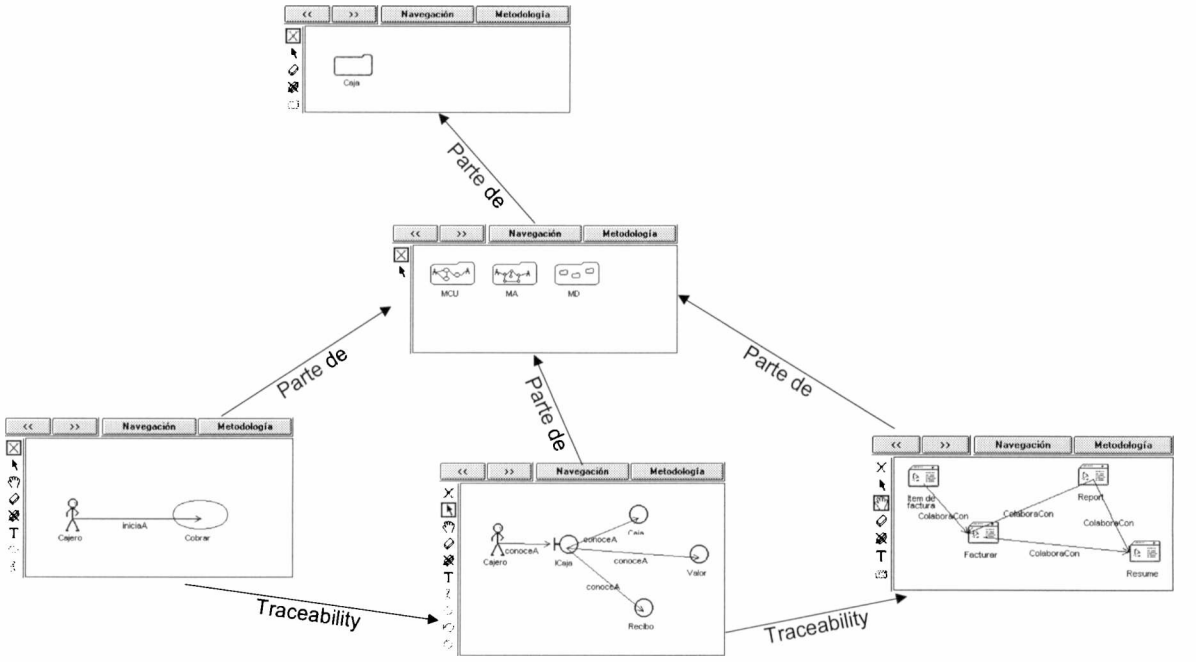


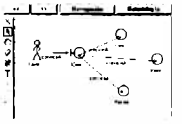
Repositorio

Arquitectura del Repositorio

En el repositorio encontramos dos tipos de elementos:

- *Documentos*: Cada elemento de un desarrollo del que vale la pena tener información debe quedar documentado, para ello se le asigna al menos un documento que administra la información del elemento. Por ejemplo Casos de uso, Actores o modelos de Test, etc.
- *Asociaciones*: Por otro lado los distintos elementos (la información que contienen) están íntimamente relacionados, HCase provee un mecanismo para modelar y documentar estas relaciones. Por ejemplo Un actor "Inicia" un caso de uso o un objeto es "Parte de" otro.





Documentos

Un *documento* es la forma de capturar la información de cualquier entidad de relevancia en el desarrollo de software. El documento de HCase posee varias responsabilidades y atributos.

El atributo mas importante es el contenido, el contenido es la parte que modela la información central del documento, en el prototipo es un texto o un gráfico pero no hay restricciones al respecto. El contenido se puede presentar al usuario, en general utilizando una herramienta o editor. En general el contenido es "Editable" esto quiere decir que el autor o quien corresponda puede introducir modificaciones en el contenido a través de un editor.

Otra característica importante es la inclusión de un conjunto de descriptores, los descriptores son atributos que se utilizan para la búsqueda y localización del documento, entre los mas importantes figuran el autor, la fecha de creación y el tipo de documento, esta es una característica que ayuda mucho a las herramientas de navegación y reuso.

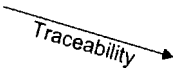
Un documento, también maneja un conjunto de asociaciones, las asociaciones son objetos que modelan las dependencias entre documentos. El documento puede agregarlas, cortarlas, clasificarlas, retornarlas y verificar su corrección.

Un documento puede pertenecer a cualquier etapa del ciclo de desarrollo, al no haber restricciones sobre el contenido, puede ser una clase, un caso de test, etc.

Puede ser compartido por varios proyectos y, para algunos predicados, puede responder si lo satisface o no lo satisface.

En base al análisis hecho un documento puede tomar varios roles según el punto de vista que le apliquemos, los roles mas destacados son Nodo de grafo, Nodo de hipermendia, Documento, Entidad "Nodo" de una base de datos, Elemento del dominio de un sistema de lógica de primer orden.

Asociaciones



Es aceptado a nivel general que la mente humana opera sobre la base de asociaciones mas que sobre una base lineal, y de acuerdo con Kerola (1985), los sistemas de información bien diseñados deben reflejar tales necesidades de procesamiento de información.

El usuario humano "asocia" naturalmente los distintos documentos que aparecen en un desarrollo, por ejemplo un caso de uso que "Extiende" a otro o una clase que "colabora" con otra.

HCase provee un soporte importante para estas *asociaciones*.

Prevé la documentación de estas asociaciones, dado que es de suma importancia en un desarrollo. Los gráficos con los distintos elementos del desarrollo y sus asociaciones son muy comunes cuando se desarrolla software. Por ejemplo el modelo de casos de uso muestra asociaciones "Extends" y "Uses" entre distintos casos de uso.

Les otorga comportamiento y responsabilidades a las asociaciones, estas pueden hacer chequeos de consistencia o manejar "anotaciones" del desarrollador.

Las utiliza como información de entrada para sus herramientas. Por ejemplo, el browser hipermedial utiliza las asociaciones como links de hipermedia, permitiendo navegar de un caso de uso al caso de uso que lo extiende o de una tarjeta de clase a la tarjeta de la clase con la que colabora. Esta es una característica muy potente, ya que se cuenta con un conjunto de links sumamente útiles, que no requieren mantenimiento y cuya consistencia esta garantizada.

Para cualquiera de los tres objetivos mencionados anteriormente, presentación, comportamiento o información para una herramienta, se utiliza un solo objeto. Una asociación sabe presentarse, posee información extra que puede ser utilizado por distintas herramientas y posee comportamiento propio, esto le otorga a HCase un potente medio para el mantenimiento de la consistencia y la integración de distintas herramientas.

Las asociaciones de HCase son binarias, dirigidas, expresables en forma inversa, tienen tipo y otros descriptores.

En base al análisis hecho un arco o asociación puede tomar varios roles según el punto de vista que le apliquemos, arco de grafo, link de hipermendia, Asociación

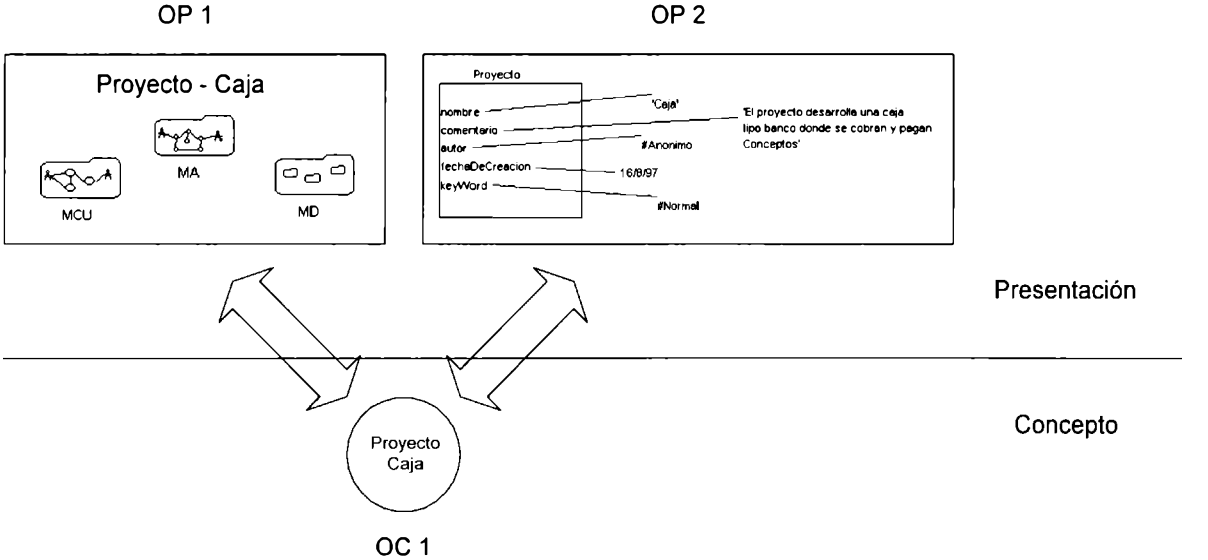
interdocumento, entidad "Arco" de base de datos o Relación binaria entre elementos del dominio.

Niveles de concepto y de presentación

Si bien tanto los documentos como las asociaciones pueden ser presentados al usuario, entendemos que es mas intuitiva la presentación de documentos. Por esta razón esta sección se explica en términos de documentos pero el lector no debe olvidar que todos los conceptos también son aplicables a asociaciones

Un documento es un par Presentacion-Concepto donde la *presentación* es un objeto que posee el conocimiento de como debe ser presentado el documento al usuario humano y el *concepto* es un objeto que mantiene la información del documento independientemente de como sea mostrada, trata de capturar la semántica del documento.

El objetivo de esta separación es la posibilidad de presentar un concepto o elemento de software desde distintos puntos de vista o perspectivas, incluso con diferentes herramientas, sin perder la consistencia. Es muy frecuente que a lo largo de un desarrollo (o varios) un mismo elemento de software aparezca reiteradas veces, por ejemplo encontraremos proyectos. Además frecuentemente encontraremos que los conceptos representan elementos del desarrollo como clases, métodos y atributos, el concepto es el encargado de mantener la referencia al elemento en cuestión y tiene como responsabilidad hacer de interfaz entre el documento que refleja la información del elemento y el elemento propiamente dicho. De esta forma pretendemos controlar la consistencia entre el modelo y el sistema desarrollado.



Por cada proyecto tendremos un concepto que contendrá el nombre del proyecto, una descripción y la información de como se relaciona con otros objetos. Por otro lado encontraremos al menos dos presentaciones: Una presentación muestra el proyecto como un gráfico que contiene figuras representando los diferentes modelos la otra presentación muestra otras propiedades del proyecto en un gráfico completamente

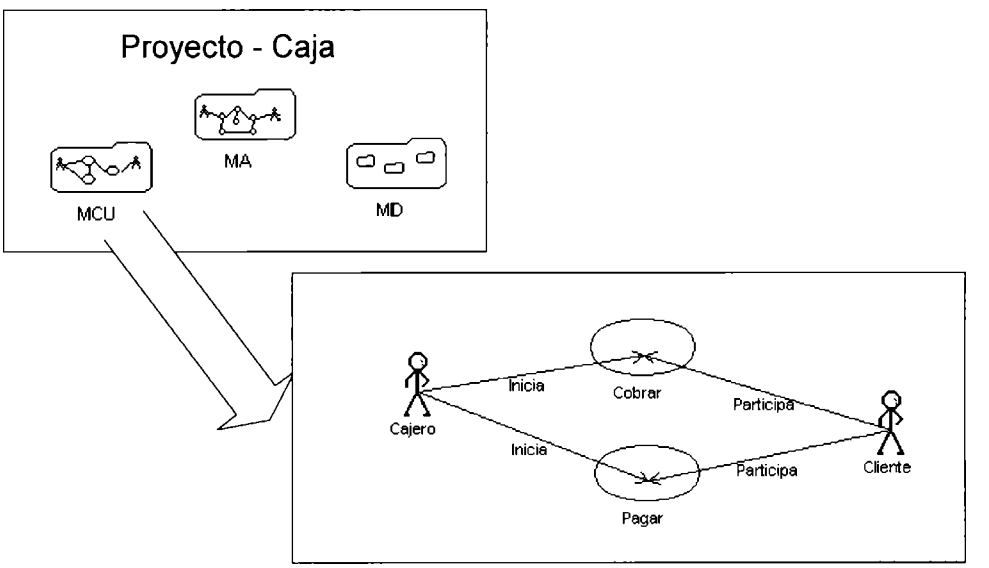
diferente. Las dos presentaciones obtienen la información del concepto lo que garantiza la consistencia.

Uno de los requerimientos mas importantes de las herramientas CASE es la posibilidad de presentar los elementos desde distintos niveles de abstracción.

Desde esta base nos encontramos con que los documentos de niveles de abstracción altos normalmente se componen de elementos que, en un nivel de abstracción mas bajo, se perciben como verdaderos documentos. De esta manera la presentación del documento de nivel mas alto se compone de elementos que son presentación de otros conceptos formando un *documento compuesto*. Un caso típico es un diagrama de colaboraciones donde cada tarjeta es la presentación de una tarjeta, la cual es un documento en si misma.

De lo anterior se desprende que no todo par presentacion-concepto forma un documento, es muy frecuente que sea solo parte de otro documento que los contiene, así como una tarjeta es parte del diagrama de colaboraciones.

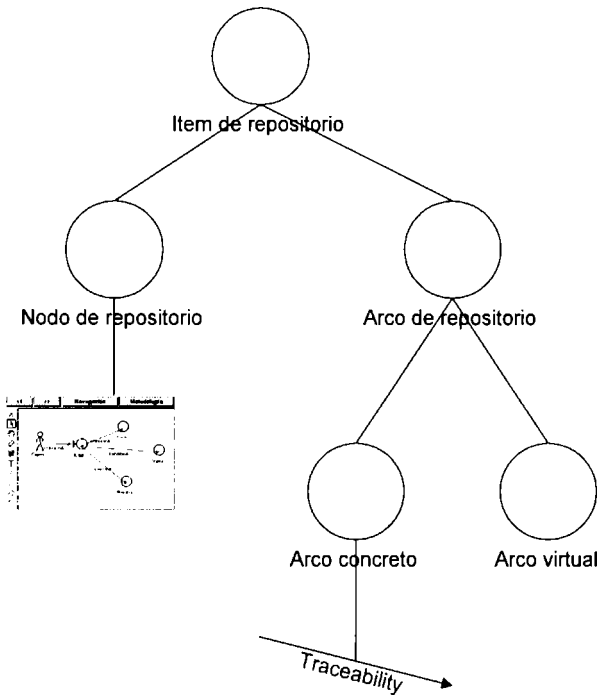
En el ejemplo vemos dos presentaciones, la superior es una de las presentaciones del proyecto Caja, una de las tres figuras que componen este documento es presentación del modelo de casos de uso. La inferior es una presentación del modelo de casos de uso (MCU). La ventaja de esta aproximación es el mantenimiento de la consistencia ya que cualquier cambio en el modelo de casos de uso se vera reflejado en ambos documentos.



Una vez que una presentación es asociado con un concepto, la relación permanece por toda la vida de la presentación.

Nivel de Conceptos

El diseño de los conceptos de HCase se basa en una pequeña jerarquía de clases muy general, a partir de ellas se especializan conceptos que capturan la semántica de los distintos documentos y asociaciones que aparecen en el repositorio. La jerarquía consiste de:



Todos los conceptos que se encuentren en el repositorio (nodos o arcos) heredan de *item de repositorio*, esta clase abstracta satisface las responsabilidades de creación, eliminación, manejo las presentaciones, mantenimiento de descriptores, manejo de tipos y sienta las bases para la implementación del grafo, el sistema lógico y la base de datos.

El *nodo* es uno de los herederos de ítem de repositorio, cumple el rol de concepto en los pares concepto-presentacion que forman documentos o partes de ellos. Maneja las responsabilidades que tienen que ver con el contenido y las asociaciones, además especializa la mayoría del comportamiento del ítem de repositorio.

Los *arcos* conectan nodos, cumplen el rol de concepto en los pares concepto-presentación que forman asociaciones.

Son binarios. Conectan exactamente 2 nodos.

Son orientados. Uno de los nodos conectados es el origen y el otro es el destino.

Tienen inversa. Se puede invertir la orientación, invirtiendo la semántica de la asociación, por ejemplo si una clase A "hereda de" una clase B => B es "heredada por" A.

Los arcos manejan las responsabilidades de conocimiento de los nodos que asocian, orientación de la asociación e inversa. Al igual que los nodos especializan la mayor parte del comportamiento de los ítems de repositorio.

Se previo la existencia de *arcos virtuales*, estos arcos se implementaron para el caso que una herramienta de navegación pueda guardar copia de los arcos navegados, la gran diferencia entre estos arcos y los arcos comunes es que su creación y eliminación no tienen impacto en el repositorio.

Dentro del diseño de los nodos se le dio un tratamiento especial a los arcos tipo "*Parte de*" o "*Consiste en*", dado que es una asociación usada prácticamente en todas las

metodologías y que puede ser muy útil para que los usuarios no se pierdan y da una sensación de ubicación.

Otro detalle destacable es la creación de clases de nodos especiales, entre ellas se destacan:

Nodos Proyecto

Nodos Usuario

Nodos repositorio de proyectos

Presentación

Los diagramas y gráficos son herramientas muy importantes para la modelización, un diagrama o gráfico puede transmitir al lector aspectos de los modelos que serian muy difíciles o imposibles de transmitir mediante el texto.

Los diagramas y figuras muchas veces son representaciones de componentes que luego encontramos en los distintos desarrollos, por ejemplo clases, subsistemas, atributos o responsabilidades.

Una tarea importante pensada para HCase es la posibilidad de crear gráficos, pero que los gráficos vayan un poco mas allá y tengan comportamiento y se conecten con los elementos del desarrollo que representan, de esta manera cambios en cualquiera de los dos pueden desencadenar actualizaciones o advertencias de los chequeos de consistencia.

Las presentaciones son los objetos que permiten a los conceptos presentarse en un diagrama o gráfico en forma consistente. La idea es que la presentación tenga responsabilidades propias que son requeridas por un editor para mostrar un diagrama, la posición, tipo de letra, colores, dimensiones, etc. Mientras que la información conceptual sea adquirida del concepto que presenta.

Por cada concepto de la metodología tenemos una o varias presentaciones, lo que permite manejar con flexibilidad el aspecto visual de acuerdo al espectador o según el contexto.

Las presentaciones pueden proyectarse en diferentes medios tales como gráfico, texto, vídeo y audio lo cual permite que HCase sea un poderoso medio de información hipermidial.

Las presentaciones mantienen un constante dialogo con su correspondiente concepto, reaccionando a cada evento de manera de mantenerse siempre consistentes. Este dialogo es sumamente complicado ya que la presentación obra de interfaz entre el usuario y el concepto.

Para manejar las presentaciones se especializaron dos clases centrales del FrameWork HotDraw, HFigure de CompositeFigure, y HDrawing de Drawing,. Las presentaciones de HCase son herederas de alguna de estas dos nuevas clases.

Herramientas

Además de la información contenida en los documentos y las asociaciones (Conceptos + Presentaciones) se debe proveer herramientas para acceder y actualizar la información. Esto es logrado a través de browsers, editores y otras herramientas.

El prototipo de HCase cuenta con algunas herramientas, un browser hipermedial y un algunas herramientas de apoyo para la navegación, además se integraron algunas herramientas del ambiente, pero lo mas importante es que por su diseño abierto permite integrar cualquier otro editor o herramienta que pueda ser desarrolladas. Esto hace de HCase una herramienta potencialmente poderosa. Se han pensado para HCase herramientas de chequeo de consistencia, browsers alternativos, mapas de navegación, herramientas de reuso, etc.

La herramienta que se provee como interfaz de HCase es un browser hipermedial (HBrowser) que permite realizar las actividades básicas de generación, exploración y mantenimiento de los documentos y asociaciones que forman los proyectos.

HCase esta diseñado para tener interfaz hipermedial, los conceptos básicos de hipermedia fueron diseñados de la siguiente manera:

Un *nodo* es un contenedor de información el cual provee una representación para estructurar elementos de información (Documento).

Un *link* especifica la conexión semántica entre dos nodos (Asociación).

Un *ancla* designa la fuente o destino de un link, pudiendo ser un nodo o parte de un nodo.

Un *hiperdocumento* es una colección de nodos con sus respectivos links (Proyecto).

Un *hiperespacio* es una colección de hiperdocumentos y sus links. Hiperespacios colaborativos son hiperespacios compartidos por personas en un equipo o una organización (HCase).

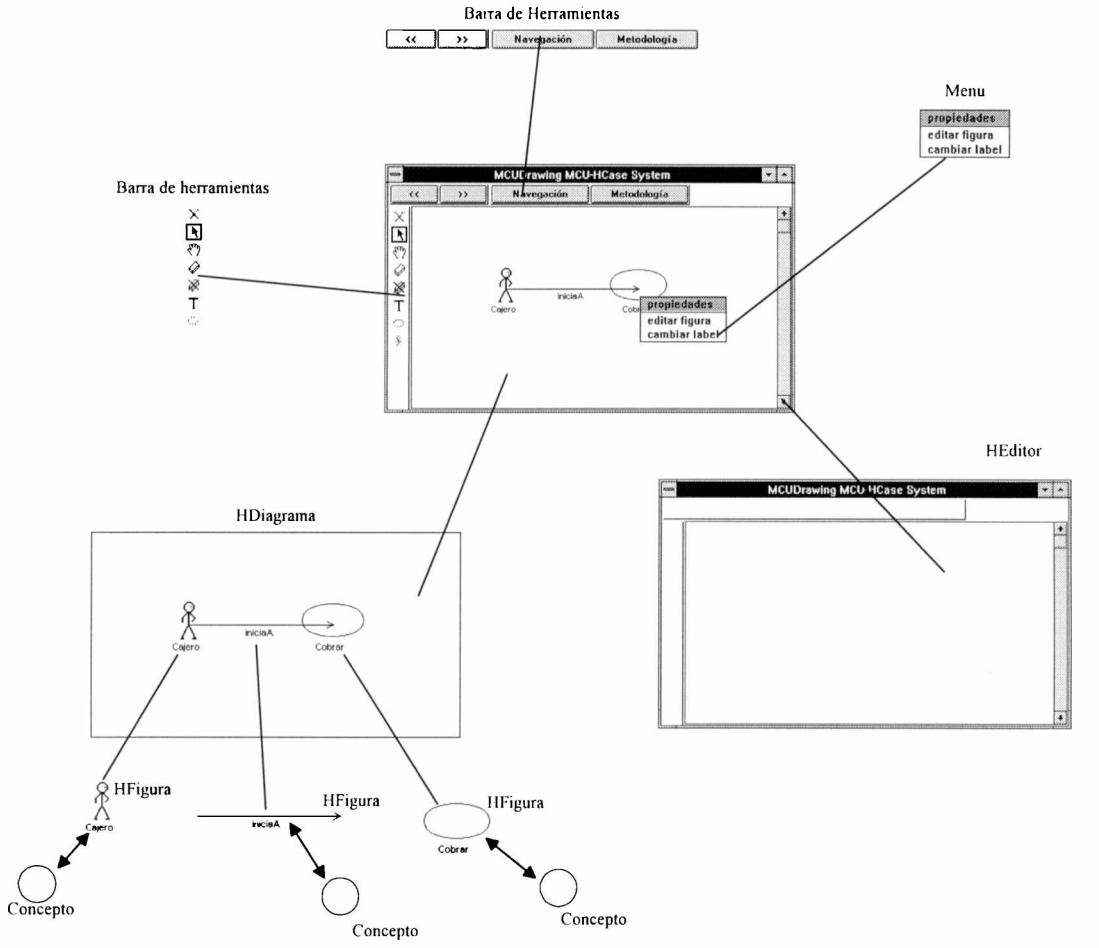
Los *autores* son los que crean las componentes de hipertexto y los *lectores* son los que "navegan" a través de la información.

HBrowser es una especialización de la clase DrawingEditor del Framework HotDraw.

En HBrowser los nodos son pares presentación-concepto donde el concepto es un nodo del repositorio y la presentación puede ser simple o compuesta, mientras que los links son los arcos que asocian conceptos.

Diseño de HBrowser

Presentación y edición de diagramas

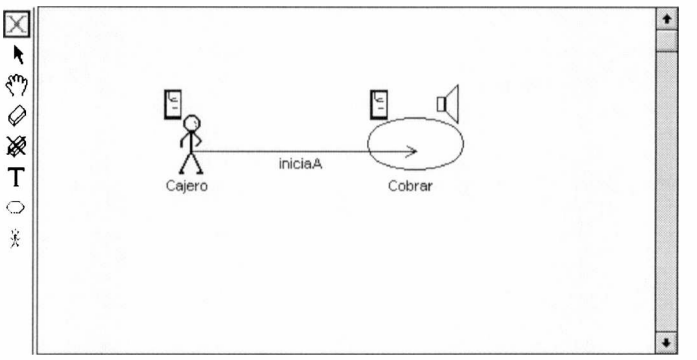


Estructura

Para poder describir mejor la arquitectura de HBrowser inicialmente dejaremos de lado toda la funcionalidad asociada a la semántica y al contexto de los diagramas diagrama con respecto al desarrollo de software del que son parte. Sin la funcionalidad asociada HBrowser puede verse como un editor de gráficos o diagramas, desde esta perspectiva podemos presentar la arquitectura de HBrowser basada en los siguientes componentes:

HEditor. HBrowser es un editor de gráficos o diagramas que principalmente tiene las responsabilidades de presentar el diagrama al usuario y capturar las acciones del mismo convirtiéndolas en operaciones sobre el diagrama.

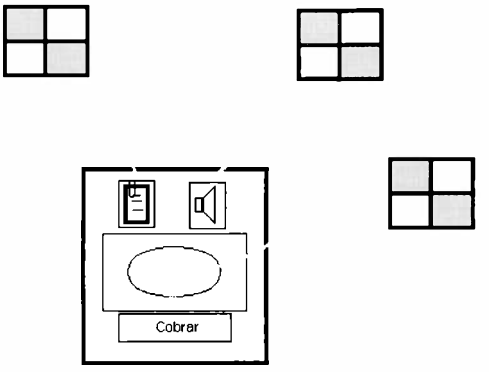
HDiagrama. Es un objeto que posee todo el conocimiento referido al diagrama o gráfico que debe presentar HEditor. Sus responsabilidades principales tienen que ver con brindar a HEditor toda la información que este requiera para poder presentarlo y traducir en acciones todas las operaciones que HEditor interprete de las acciones del usuario.



HDiagrama modela un documento de la metodología y como parte de sus objetivos tiene el de proveer un marco sintáctico donde el usuario trabaja, esto tiene que ver con la responsabilidad de traducir en acciones las operaciones del usuario, HDiagrama permitirá hacer solo operaciones que mantengan al diagrama consistente con las reglas sintácticas que defina la metodología.

HFigura. Cada diagrama esta compuesto por figuras, estas forman parte del diagrama y poseen responsabilidades parecidas. Fundamentalmente HDiagrama maneja las responsabilidades del conjunto y delega a las HFiguras las responsabilidades puntuales.

Las HFiguras están compuestas por figuras mas simples y una de sus responsabilidades mas importantes, además de ser elementos sintácticos de la metodología, es la de servir como área sensible y colaborar en la navegación.



Concepto. Cada HDiagrama y cada HFigura esta asociado con un concepto. Esta sociedad Concepto-HFigura o Concepto-HDiagrama puede ser generalizada como una sociedad Concepto-Presentación donde el rol de Presentación puede ser jugado tanto por un HFigura como por un HDiagrama. Un concepto es un objeto que funciona como un auxiliar de la Presentación, manejando algunas de sus responsabilidades. La idea es que el concepto maneja las responsabilidades que tienen que ver con la semántica y maneje la relación con el resto del desarrollo y que la presentación maneje las

responsabilidades de interacción con el editor. El objetivo es poder presentar consistentemente un elemento del desarrollo en varios diagramas.

Menúes, Handles y Barras de Herramientas. HEditor captura la acciones del usuario mediante estos tres elementos. Los menúes son sensibles al lugar del diagrama donde se desplieguen y pueden desencadenar acciones sobre el diagrama completo u sobre una figura en particular. Las barras de herramientas son formas mas cómodas de ejecutar opciones de menú. Algunos ejemplos de herramientas soportadas por HCase son:

1. Herramienta de navegación: permite al espectador navegar al nodo al cual el área sensible está anclado.
2. Herramienta de selección: permite al espectador seleccionar un elemento gráfico, para moverlo, conectarlo a otro, cambiar sus dimensiones, etc.
3. Herramienta de borrado: permite al espectador eliminar un elemento gráfico sin removerlo del repositorio.
4. Herramienta de remover: borra el elemento gráfico y remueve el nodo de información del repositorio.
5. Herramienta de creación de Use Case: Permite al espectador crear un Use Case.

Herramienta de creación de Actor: Permite al espectador crear un Actor.

Los handles son mecanismos que permiten realizar operaciones puntuales sobre algunas figuras.

Operaciones

Cuando abrimos un diagrama, no solo podemos ver la información que contiene sino que además permite editarla.

En el punto anterior analizamos la estructura interna de un HDiagrama, la edición produce cambios en el estado interno de cualquiera de las componentes, las operaciones de edición desencadenan el envío de un conjunto de mensajes que producen automáticamente los cambios correspondientes en las componentes apropiadas.

Crear figuras hard

Eliminar figuras hard

Activar un Handle

Activar una opción de menú general

Activar una opción de menú especifico de una figura

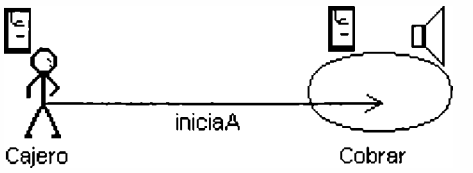
Utilizar una herramienta

Estructura de HFigura

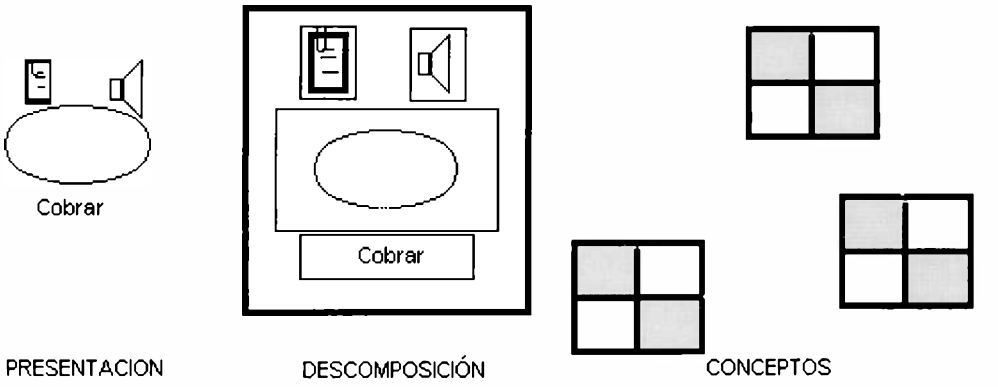
Las HFiguras se componen de figuras menos complejas que muestran diferentes propiedades y relaciones del concepto, cada una de las cuales se comportan como área sensible ancladas a nodos dentro del repositorio de objetos.

Elemento gráfico de la metodología: Una de las figuras que componen a la PS se denomina *figura principal* (figura que describe el use case) la cual forma parte de la sintaxis de la metodología, dependiendo del contexto las PS forman parte de gráficos

semánticos más complejos. En este tipo de gráficos generalmente las PS se encuentra asociadas con otras. Las asociaciones también son PS de arcos en el repositorio.



En la siguiente figura se muestra la HFigura de un Use Case, su composición y el concepto presentado.



Creación de nuevos diagramas

HBrowser no posee una opción directa para crear un nuevo diagrama, esta decisión se basa en dos argumentos.

Una de las razones es la decisión que se tomo de hacer una estructura básica de documentos unos "Contenidos" dentro de otros, para esto se requiere conocer el "contenedor" de cada documento en el momento de la creación por eso se decidió crearlo desde el contenedor. Mas adelante ampliaremos este concepto.

La otra razón es brindar soporte al usuario para aplicar la metodología, en este sentido se definió de antemano una estructura básica para el proyecto y se establecieron reglas para crear y eliminar documentos automáticamente a medida que el usuario avanza con el proyecto en este sentido desarrollaremos un ejemplo.

Por las dos razones anteriores se diseñó, en HBrowser, una forma indirecta de crear diagramas, esta forma se podría enuncia de la siguiente manera.

Durante la edición de diagramas se producen eventos que crean nuevas figuras, eliminan otras y modifican diversas propiedades del diagrama, ante determinados eventos HBrowser reacciona creando nuevos Diagramas, decimos que estos últimos fueron creados en forma indirecta, además de los diagramas se crean todo tipo de componentes como conceptos, arcos, figuras, etc.

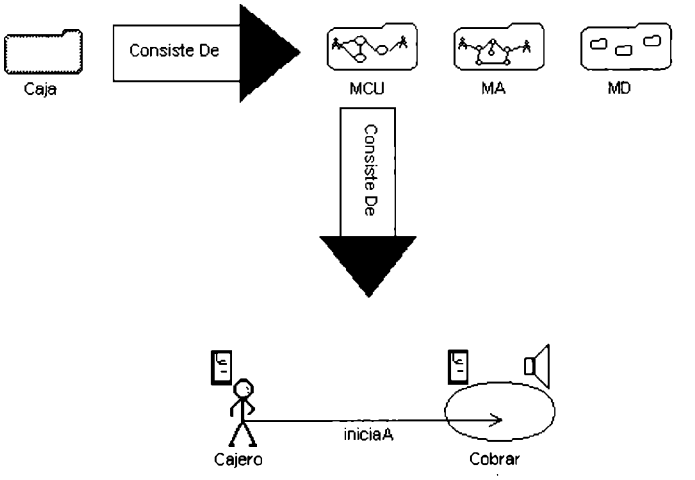
Veamos dos ejemplos:

1. La forma de crear un caso de uso es navegar hasta el modelo de casos de uso y utilizando la herramienta apropiada crear una figura de caso de uso. Al crear la figura también se crea un diagrama que contiene la información del nuevo caso de uso y una asociación "Parte de" entre el modelo de casos de uso y el caso de uso. La figura creada también es un "ancor" que permite navegar hacia el nuevo documento creado.

2. Un evento puede desencadenar la creación de varios objetos , por ejemplo la creación de un proyecto dispara la creación automática de los diagramas vacíos que necesariamente deberán ser incluidos en el desarrollo y las asociaciones entre ellos, (un diagrama para realizar la especificación de requerimientos, otro para el modelo de análisis, uno para el modelo de diseño, otro para reflejar la estructura del proyecto y las asociaciones "Parte de" correspondientes) Esto genera un esqueleto donde construir el proyecto y con el browser hipermedial se obtiene un recorrido básico de los documentos.

Jerarquía de contenedores

Una de las restricciones mas importantes impuestas a los documentos de HCase es que todos sus documentos deben estar incluidos en la jerarquía de contenedores. La jerarquía de contenedores es una estructura de datos en forma de árbol cuya raíz es el *repositorio de proyectos* y que se extiende a través de la relación "Consiste de", reflejando una jerarquía de documentos contenidos, recursivamente, unos dentro de otros. El repositorio de proyectos "Consiste de" un conjunto de proyectos, cada proyecto "Consiste de" un conjunto de modelos y así sucesivamente los conceptos se contienen unos a otros.



Dado un documento, HBrowser, permite navegar hacia cualquiera de sus contenidos, la navegación hacia el contenedor presenta otras dificultades, ya que puede haber un documento que tenga mas de un contenedor, en ese caso se selecciona uno a criterio del usuario. Esta forma de navegación es muy intuitiva y una ayuda importante para la navegación ya que brinda un criterio potente para

la búsqueda de un documento en el contexto general, además permite moverse por distintos niveles de abstracción.

Para el caso del ejemplo la navegación seria desde en nodo "Repositorio de proyectos" al nodo "Documentos del proyecto: Máquina recicladora"

Navegación

Para explicar la forma en que se implementa la navegación usando HBrowser tenemos que volver un poco sobre la estructura de un nodo. Un nodo esta compuesto por 2 objetos que colaboran ente si cumpliendo una parte de las responsabilidades del nodo, el diagrama o presentación y el concepto. El diagrama tiene las responsabilidades que tienen que ver con la presentación del diagrama y la tarea de edición, el concepto cumple con la responsabilidad que tiene que ver con la semántica o concepto que representa el diagrama y con la forma en que ese concepto se combina con otros conceptos para construir un modelo complejo.

Podemos pensar que los links tienen que ver con asociaciones que una persona, que esta navegando, hace entre los distintos conceptos. Creemos que solo excepcionalmente la persona tratara de asociar dos nodos por algo que tenga que ver con su presentación. Por esta razón decidimos hubiera el conocimiento relacionado con los links a nivel de conceptos. Al modelar las asociaciones de esta manera obtuvimos una estructura de grafo donde los nodos son los conceptos y los arcos son los links o asociaciones entre conceptos.

En base a lo explicado y agregando que un concepto puede tener asociado una o mas presentaciones o diagramas, se puede especificar la primer forma que seleccionamos para soportar la navegación:

Dado un nodo obtenemos del concepto todos los arcos que salen de el.

Seleccionando uno de estos arcos obtenemos un concepto asociado al que deseamos navegar.

Una vez que obtenemos el concepto seleccionamos una de las presentaciones o diagramas para navegar.

Seleccionado el nuevo diagrama sacamos del editor el diagrama que estabamos presentando e instalamos el nuevo diagrama.

Esta forma de navegación permite explotar mediante una interfaz hipermedial la riqueza expresiva de la metodología, cuando mas rica sea la metodología mayor cantidad de opciones de navegación tendrá Relaciones de la metodologías

Como segunda forma de navegación importante tenemos que mencionar a las distintas figuras que aparecen en los diagramas, anteriormente dijimos que cada una estaba asociada a un concepto en forma similar al diagrama y también es muy frecuente que funcionen como áreas sensibles que permiten navegar a otros nodos. Esta forma de navegar es una simplificación de la primera ya que la ya que se hace automáticamente la selección del arco: Es el arco "Parte de" que existe entre el concepto asociado al diagrama y el concepto asociado a la figura. En este caso estaríamos explotando la Jerarquía de contenedores que se establece entre los distintos conceptos de HCase,

Historia de nodos visitados

Existe objeto que colabora en forma cercana con el editor y que es el Manager de navegación. La responsabilidad de este objeto es registrar en un "log" todos los eventos de navegación que se produzcan en HBrowser.

La estructura interna del manager esta basada en una pila. Cada vez que HBrowser navega el manager de navegación registra el nodo destino el arco navegado y el nodo origen de manera de poder reproducir cualquier paso de la navegación.

El manager de navegación ha sido dotado de una interfaz que permite al usuario operar sobre la pila de nodos y arcos utilizados. El manager de navegación es una herramienta fundamental para la efectividad de HCase y uno de las entidades para la que previmos la incorporación de mucha funcionalidad.

Herramienta de reglas

La herramienta de reglas permite controlar la forma en que los conceptos son asociados. En los puntos anteriores describimos la parte conceptual del repositorio como un grafo donde los nodos son conceptos y los arcos son asociaciones entre conceptos. Existen varios tipos de nodos y varios tipos de arcos, en general representan los distintos conceptos y relaciones especificados en la metodología de desarrollo modelada. Uno de los conocimientos que debe ser modelado para brindar un soporte adecuado de la metodología es el conocimiento de que tipos de conceptos pueden asociarse y con que asociaciones.

Veamos algunos ejemplos de la metodología de Jacobson:

Para manejar estas reglas existe una herramienta que permite al usuario especificar las reglas. Este objeto maneja una colección de reglas de la forma Tipo de nodo Origen, Tipo de arco, Tipo de nodo destino y posee una interfaz para que el usuario pueda presentar distintas visiones de la colección.

Por otra parte cada vez que se crea un arco se verifica en esta estructura que satisfaga alguna de las reglas.

Impacto de las extensiones previstas

Se muestra el diseño preliminar de las herramientas previstas y con un análisis del impacto que produciría su construcción (O mostrando la forma en que el repositorio se la banca) .

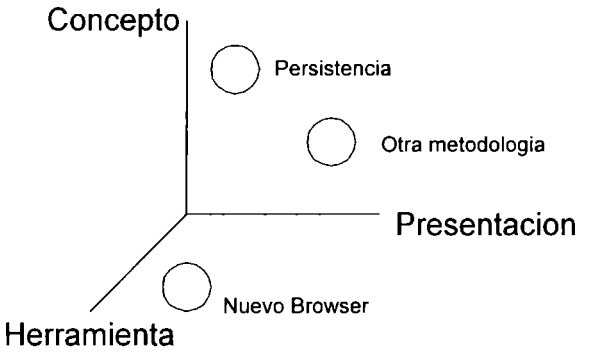
Métrica de extensibilidad

HCase se desarrollo a través de tres "Dimensiones", por un lado encontramos la dimensión conceptual o de repositorio, en esta dimensión se pueden hacer extensiones como persistencia, nuevas validaciones y chequeos de consistencia.

Por otro lado se desarrolla la dimensión de presentación en esta dimensión encontramos extensiones como presentaciones alternativas, propiedades extra en los gráficos, comportamiento mas inteligente de las presentaciones.

Como tercer dimensión encontramos a las diversas herramientas que se integran en HCase, en esta dimensión encontramos extensiones como nuevas herramientas o incremento en la potencia de las ya existentes, por ejemplo la potenciación del comportamiento Hipertextual.

De acuerdo al tipo de extensión afectara solo una, dos o las tres dimensiones, la complejidad de la extensión es directamente proporcional a la cantidad de dimensiones que afecta, esto nos da una buena medida de la extensibilidad de HCase.



Estrategias para extender

El repositorio

La estrategia de separar el repositorio de las herramientas es una de las mas importantes. El concepto es muy simple Sean H1, H2, H3, y R 3 Herramientas y el repositorio.

Hacer un cambio en el repositorio tiene Complejidad 1 ya que no afecta a ninguna herramienta, hacer un cambio en una herramienta o agregar una nueva también tiene complejidad 1 ya que no se ve afectado el repositorio, en el peor de los casos una nueva herramienta nueva o la modificación de una existente requerirá nueva funcionalidad en el repositorio, en este caso la complejidad es 2, es decir que la mitad de la herramienta no se ve afectada por el cambio.

Cuanto mayor sea el numero de herramientas mejor será el resultado obtenido. No se debe dar el caso que un cambio en el repositorio afecte a las herramientas ya que

la funcionalidad del repositorio no debería verse modificada, a lo sumo su estructura interna o el agregado de nueva funcionalidad.

El FrameWork

Otra estrategia importante para lograr la extensibilidad de HCase es la utilización de un FrameWork para la construcción de HBrowser. Desde esta perspectiva el objetivo es que las extensiones a la funcionalidad de HBrowser sean extensiones al FrameWork.

Detalle de extensiones previstas

En esta sección se presenta, en algunos casos, un diseño preliminar de las extensiones previstas, y en otros un breve análisis del costo e impacto de realizar estas extensiones.

Otros browsers

La primer conclusión que podemos sacar es que incluir un nuevo browser no es una tarea compleja, mas allá de la complejidad propia del browser. Aun si la inclusión del nuevo browser implicara extensiones al repositorio HBrowser se podría mantener totalmente excluido del nuevo Browser.

Dos de los nuevos browser que se nos ocurrieron fueron el de una jerarquía facetada y uno que haga querys a la base de datos.

En estos dos casos alcanza con algunas extensiones al grafo y la construcción del nuevo browser.

Una de las preguntas que surgen es que pasaría si el nuevo browser es hipermedial y también asocia presentaciones, que sean o no HDiagramas, a conceptos en este surgen varios inconvenientes pero que ya han sido previstos en el repositorio.

En primer lugar surge un inconveniente al navegar por los arcos del repositorio, como un concepto puede tener varias presentaciones hay que implementar un algoritmo para seleccionar la presentación a instalar en el browser.

Por otro lado hay que prever un esquema de actualización del estilo "Observer" para mantener consistentes y actualizadas todas las presentaciones con respecto al concepto, por ultimo hay que prever un esquema de búsqueda para poder buscar un concepto ya existente y asociarle otra presentación.

Enriquecer la funcionalidad de los documentos existentes

Dado que es un prototipo no fue modelado en forma completa toda la metodología, algunos conceptos importantes podrían ser incorporados, por ejemplo permitir tener varios modelos del mismo tipo por proyecto y soportar traceability entre los modelos.

Navegar documentos externos - Editor del contenido

Los documentos de HBrowser son HDiagramas, pero además de diagramas y figuras es necesario incluir en los desarrollos otro tipo de objetos como grabaciones de entrevistas, Clases de Smalltalk, prototipos, diseños de interfaz.

Seria poco criterioso tratar de incluir dentro de HBrowser soporte para todas las tecnologías que pueden ser utilizadas para modelar y documentar un desarrollo, sobra todo si tenemos en cuenta que ya han sido desarrolladas herramientas suficientemente poderosas para manejar estos elementos. Creemos mucho mejor permitir integrar HBrowser (y HCase) con otras herramientas.

Como ejemplos de esto elegimos dos editores que utiliza Object Works en forma habitual para sus desarrollos, uno de los ejemplos es permitir abrir desde una tarjeta de clase el "Class Browser" que permite editar el código de la clase. Esto se logra con una pequeña modificación en el menú de la tarjeta de clase. Otro ejemplo muy claro y potente es la posibilidad de abrir un editor adicional para un objeto interfaz, este editor me permite construir un prototipo de la interfaz y puede ser abierto directamente desde la figura del objeto interfaz. Concluyendo, HBrowser permite la fácil, integración de otros editores de manera de explotar la potencia de otras herramientas.

Navegar por links definidos por el espectador

Una de las ideas de proveer interfaz hipermedial para HCase es la de brindar al desarrollador la flexibilidad y la potencia de reflejar asociaciones poco estructuradas, sobre todo en las primeras etapas del desarrollo.

Para dar soporte a la idea de las asociaciones poco estructuradas no alcanza con las asociaciones entre conceptos que contiene el prototipo, es necesario proveer asociaciones libres y cuya semántica sea determinada por el usuario. Las asociaciones definidas por el usuario no tienen una semántica claramente determinada, ya que es la que interpreta el usuario.

Debido a que el objetivo del nivel mas bajo del repositorio (grafo) es reflejar una estructura que pueda ser compartida por varias herramientas los links definidos por el usuario no deben estar en este nivel.

La ubicación natural para los links definidos por el usuario es el nivel de presentación, para implementar este tipo de links debemos:

Crear un nuevo tipo de figura que funcione como ancor.

Idear una interfaz que permita seleccionar el diagrama donde nos llevara el link.

Crear el método que implemente el efecto de navegación.

Debido que HBrowser esta construido sobre un framework, la construcción de la nueva figura es perfectamente realizable. El método que permite navegar es sumamente simple y la construcción de la interfaz para seleccionar una presentación es un poco mas complicada pero es sumamente modular y no impacta sobre el resto de la aplicación.

Anotate

Este tipo de links es muy común en las herramientas hipertextuales, permite agregar un ancor que abre un cuadro de dialogo para leer y editar un comentario, este es una especialización del caso anterior donde hay que hacer la figura con capacidad para abrir el cuadro de dialogo y contener el comentario, la estructura del framework hace que esto sea muy sencillo.

Bookmark

La idea de esta funcionalidad es que el usuario "Marque" un nodo al que supone querrá volver mas tarde, de la misma manera que se marca un libro.

La forma de implementar esta funcionalidad es hacer una pequeña modificación en HBrowser para captar, por medio de un menú u otra alternativa, la intención del usuario de marcar un nodo y transmitirle esta intención al manager de navegación.

Una modificación mas importante al manager de navegación permite agregarle la funcionalidad necesaria para que el usuario pueda seleccionar un nodo marcado.

En este caso las modificaciones importantes están circunscriptas a una sola clase lo que hace que sea de baja complejidad.

Links desde el texto (HotWords)

La idea, en este caso, es permitir mediante distintos tipos de letras y otras formas de resaltar texto, indicar que una palabra o porción de texto es un ancor que permite navegar a un nodo donde hay mas información referida a esa palabra o porción de texto.

Este es un caso donde la complejidad de la modificación es importante, dado que el prototipo utiliza las herramientas estándar de edición de texto que provee Object Works es necesario tener un conocimiento mas que importante de las clases que tienen esta responsabilidad.

Luego de evaluar el costo de aprender el funcionamiento de las clases creímos que debíamos consultar a alguien mas experto en el tema pero al no poder encontrar a nadie con esa característica tuvimos que dejar sin evaluar este punto.

Soporte para contextos

Una característica que le puede dar mucha potencia a una herramienta de desarrollo es la posibilidad de funcionar de una u otra manera de acuerdo a un cierto contexto.

El contexto puede estar determinado por infinidad de variables como el proyecto que se esta recorriendo, las autorizaciones que el esquema de seguridad asigna al usuario o el rol que desempeña el usuario dentro del proyecto. Por esto se diseño un mecanismo que permite a HBrowser reaccionar según el contexto.

HDiagrama. trabaja con dos mecanismos para reaccionar según el contexto, permiso de herramientas y visualización de elementos gráficos. En el primer caso la paleta de herramienta muestra solo aquellas a las cuales el espectador tiene permiso, de esta forma no podrá usar cierto subconjunto de herramientas, en el segundo caso, filtra porciones de un gráfico compuesto dejando solo los elementos gráficos correspondientes al contexto.

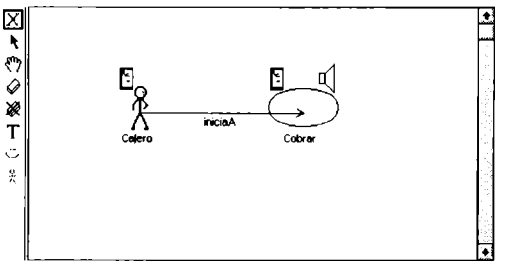


Ilustración 1

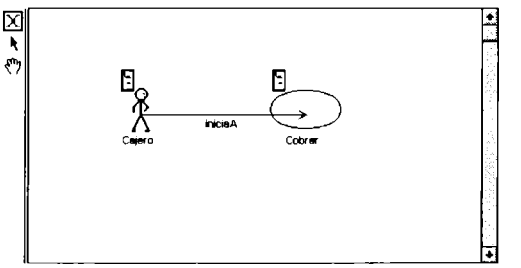
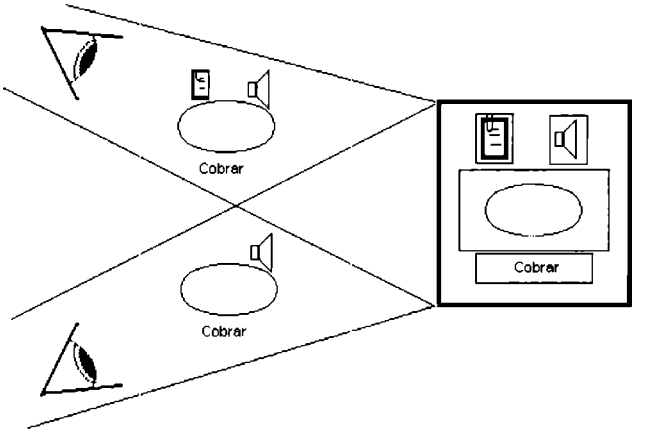


Ilustración 2

En la ilustración 1 el espectador tiene permiso para toda las herramientas de la presentación mientras que en la Ilustración 2 él solo tiene permiso a la herramienta de navegación.

Dado que la HFiguras se componen de figuras mas sencillas, también se pueden manejar por contexto y permitir comportarse en forma diferente según el contexto.



Generación automática de código

Una característica que facilita la tarea y la consistencia en un desarrollo es la generación automática de código a partir de los diagramas y demás objetos creados en el CASE.

Incorporar una característica de este tipo es una tarea compleja de por si, la ventaja que presupone HCase es la posibilidad de manejar el problema solo en el nivel del grafo.

Se pueden crear conceptos y modificar algunos existentes para que tengan una estrecha comunicación con los objetos del ambiente, que tomen la información de estos y que la actualicen según las acciones del usuario y un conjunto de reglas de derivación.

Con una estrategia de este tipo se puede utilizar la información que acumulan los conceptos para generar clases y métodos, esto sumado a la integración de las herramientas (Punto que ya hemos visto) puede posibilitar una integración total entre la documentación y la implementación y la posibilidad de utilizar el mismo ambiente en todas las etapas de desarrollo.

La facilidad que propone HCase es la de concentrar la problemática solo en una de las tres dimensiones que sugería la métrica inicial.

Chequeo de reglas de consistencia

Utilizando las facilidades que provee HCase para comportarse como un grafo o para aplicarle predicados, se pueden construir fácilmente algoritmos para chequeo de consistencia.

Como se mostró en el punto de descripción general de HCase y este tipo de chequeos se pueden implementas con pocas dificultades y obteniendo buenos resultados.

Reuso de conceptos existentes

Crear figura soft

Como tarea de edición especial hay que explicar como se reusan conceptos ya existentes (Como hago para crear el mismo actor en el modelo de casos de uso y el de análisis).

Existe una forma alternativa de crear un documento, esta forma prevé existencia del objeto concepto que corresponde al documento que se desea crear, en este caso se crea únicamente un objeto presentación y se le asocia el objeto concepto ya existente.

La creación de documentos y asociaciones nuevos no es la única alternativa para hacer un desarrollo, el reuso de elementos ya desarrollados es fundamental para la estrategia que plantea HCase. Mediante una herramienta de búsqueda se puede localizar un documento que sea de interés e incluirlo en otro documento. Un Caso típico es un Actor.

Borrar figura Soft

La eliminación de documentos y asociaciones es análoga a su creación, se puede eliminar el documento completo (presentación-concepto) o solo la presentación. Una presentación puede ser eliminada sin que ello afecte al concepto pero la eliminación del concepto implica la eliminación de todas las presentaciones.

Un concepto puede existir aunque no posea vistas.

En algunos casos la eliminación de un propietario puede desencadenar en el borrado en cadena de varios propietarios, esto depende de la metodología.

La mecánica para eliminar documentos es eliminar alguna de sus presentaciones, cada vez que se elimina una figura o un drawing se chequea si también se desea eliminar el concepto y las presentaciones asociadas.

Dar soporte a nuevos documentos

El prototipo cuenta con soporte para varios documentos y diagramas de la metodología de Jacobson, pero algunos como los diagramas de interacción o los diagramas de flujo no han sido implementado.
La necesidad de dar soporte a nuevos tipos de documentos y diagramas es uno de los problemas mas complejas de solucionar en HCase, esto es lógico si tenemos en cuenta la gran integración que existe en toda la herramienta.
Como el problema surge por la integración lo analizamos desde ese punto de vista y obtuvimos las siguientes conclusiones:

El repositorio es compartido por las herramientas y es el gran integrador de HCase, dar soporte para un nuevo documento implica, seguramente, agregar nuevos tipos de nodos y arcos que reflejen la semántica de los nuevos conceptos.

La incorporación de nuevos tipos de nodos y arcos al repositorio no es muy compleja, el inconveniente surge por el posible impacto que los nuevos tipos de nodos y arcos tengan sobre las herramientas integradas.

La solución que proponemos es una estrategia, la estrategia consiste en que los nodos y los arcos del repositorio tienen un comportamiento (conjunto de responsabilidades) que llamamos básico, la estrategia es construir las herramientas de manera que cuando se encuentren con un nodo o arco de un tipo desconocido apelen al comportamiento básico.

Por ejemplo HBrowser debe poder mostrar un diagrama básico para cualquier concepto donde se vea algunas propiedades generales a todos los nodos y se permita navegar por cualquier arco que entre o salga de el.

De esta manera se puede absorber gradualmente el impacto que produzca cualquier agregado al repositorio.

El caso particular de HBrowser

Estos cambios impactan en los dos niveles de HBrowser, repositorio e interfaz. esto significa que tanto para documentos y asociaciones la adaptación requiere la creación de nuevos conceptos y presentaciones.

A nivel de repositorio, la creación de nuevos conceptos implica la especificación de la información que el nuevo concepto maneja, y el comportamiento que le corresponde de acuerdo con la nueva metodología. Existe una sutil diferencia entre los conceptos que representan documentos y los que representan asociaciones, para los primeros debe especificarse también las formas en la que el concepto se puede asociar con otros conceptos.

A nivel de interfaz debe especificarse las diferentes presentaciones que puede tener el concepto y el comportamiento que mostrara cada presentación.

La adaptación no debe impactar demasiado en HEditor salvo un requerimiento especifico de la metodología que requiera un comportamiento especial de parte de el.

Para el soporte de esta actividad se diseño y desarrollo un conjunto de herramientas.

En el mejor de los casos significa extender un poco el dominio de los diagramas en el peor puede significar tener que utilizar herramientas existentes y no compatibles o implementar algoritmos y diagramas complejos.

En general deseamos que sea definir la nueva sintaxis (Diagramas y figuras), los nuevos conceptos (Especializaciones de los existentes), nuevas asociaciones, algunos algoritmos de chequeo y nuevas herramientas, todo perfectamente soportado dentro del framework.

Dar soporte a nuevas metodologías

Este tipo de extensiones puede presentarse como la inclusión de soporte y correcta integración de los documentos generados por la aplicación de la metodología en el marco de los ya existentes.

Es sin duda la adaptación mas compleja que puede hacerse a HCase. La forma de realizarla es repetir el caso de la inclusión de un nuevo tipo de documento tantas veces como sea necesario hasta lograr contener toda la documentación y reglas de la metodología.

De todas maneras creemos que la complejidad intrínseca de una modificación de este tipo es muy grande y que la estructura de HCase es una alternativa mas que valida para absorber esa complejidad. Este tipo de modificaciones seria impensable en otro tipo de herramientas.

Notas sobre HCase

El framework HotDraw

Creemos de utilidad dar algunas definiciones explicaciones breves sobre el framework que utilizamos como herramienta principal en el desarrollo del browser hipermedial y del nivel de presentación.

Framework: Un framework es un diseño reusable de un programa o parte de un programa expresado como un conjunto de clases [Deutch) [Johnson and Foote). Como cualquier software el esta compuesto por clases concretas y abstractas. Frameworks están diseñados por expertos en un dominio particular de aplicaciones y son usados por programadores no expertos en tal dominio.

HotDraw es un framework para construir editores de gráficos semánticos. El framework maneja varios conceptos. Un resumen de los mas importantes se desarrolla a continuación:

Editores: HotDraw puede ser utilizado para construir editores para gráficos como diagramas esquemáticos, pre-proyectos, música o diseños de programas. Los elementos de los gráficos pueden tener restricciones entre ellos, pueden reaccionar a comandos del usuario y pueden ser animados. Los editores pueden ser una aplicación completa o una pequeña parte de un sistema mas grande.

Drawing: el drawing es el contenedor de las figuras del gráfico propiamente dicho. Un drawing puede ser parte de interfaz de usuario más compleja.

Figure: Cada elemento en una aplicación HotDraw es subclase de "Figure". Hay una variedad infinita de figuras primitivas que pueden ser incluidas un gráfico, para lo cual HotDraw prevé una manera de crear nuevas figuras para cada aplicación.

Hay al menos tres formas de editar los atributos de una figura, con un "handle", con el menú "pop-up" de la figura o con una herramienta especial. Cada técnica es apropiada en diferentes casos.

Es posible componer figuras de figuras mas simples.

Es posible expresar atributos de figuras como función de otras figuras.

Handles: los handles controlan el aspecto visual de las Figure y pueden variar en comportamiento, se pueden usar para cambiar el tamaño de una figura, cambiar su color o para conectar figuras. En general, los handles pueden utilizarse para cualquier operación. Los handles pueden asociarse a cualquier parte de las figuras y pueden moverse con ellas. El uso más importante que tiene los handles es el de conectar figure. A veces esas conexiones no tienen semántica y borrarlas o moverlas no afectan a las otras figuras. Sin embargo a veces la creación de una línea produce el agregado de una restricción a otra figura o la creación de una figura nueva.

Tool: Las tools representan el modo que el usuario interactua con un gráfico. Seleccionar una herramienta de la paleta le permite al usuario manipular figuras, crear figuras nuevas, realizar operaciones sobre una figura o sobre el gráfico completo.

Presentaciones y conceptos

Cada HDiagrama y cada HFigura esta asociado con un concepto. Esta sociedad Concepto-HFigura o Concepto-HDiagrama puede ser generalizada como una sociedad Concepto-Presentación donde el rol de Presentación puede ser jugado tanto por un HFigura como por un HDiagrama.

Un concepto es un objeto que funciona como un auxiliar de la Presentación, manejando algunas de sus responsabilidades. La idea es que el concepto maneja las responsabilidades que tienen que ver con la semántica y maneje la relación con el resto del desarrollo y que la presentación maneje las responsabilidades de interacción con el editor.

El objetivo es poder presentar consistentemente un elemento del desarrollo en varios diagramas.

El párrafo anterior es básicamente la definición de Concepto en la explicación de la estructura del HBrowser. Esta definición contempla solo una parte del tema, el cual tiene un objetivo mas general y presente algunos problemas que debieron ser solucionados.

Como primer comentario podemos generalizar el objetivo de esta separación entre presentación y concepto.

En la definición se hablo de la posibilidad de presentar consistentemente un mismo concepto en varios diagramas diferentes, pero es mucho mas importante que un concepto generado durante un desarrollo pueda ser consistentemente manejado por TODAS las herramientas utilizadas en el desarrollo. Entonces el objetivo del concepto es capturar la información que puede ser de utilidad para la mayoría de las herramientas y dejar la presentación del concepto para la herramienta permitiendo dividir las responsabilidades del objeto. Seria difícil adjudicarle a una clase la responsabilidad de responder a cuanto editor desee mostrar algún aspecto y además funcionar como clase, para esto se crea un nodo en el grafo que hace de interfaz entre las herramientas y la clase y se define en cada herramienta la forma en que la clase es presentada.

El primer inconveniente que surgió desde esta nueva visión es que un concepto puede estar asociado con varias presentaciones generando una relación parecida a la que el Modelo tiene con las View en el Framework MVC. A partir de este esquema surge un problema de actualización entre las presentaciones y el concepto para que se mantengan consistentes, la solución es análoga a la dada en ese framework. (Ver Observer)

El segundo inconveniente que surgió apareció con el esquema de navegación. Los arcos del grafo conectan nodos, Al navegar utilizando los arcos del grafo podemos obtener, a partir del arco seleccionado el nodo donde vamos a navegar, pero al permitir tener varias presentaciones asociadas a un concepto surge la necesidad de elegir a cual de todas las posibles presentaciones se desea navegar. Para el prototipo alcanzo con un algoritmo simple pero a medida que se incorporan herramientas debe tenerse en cuenta este tema.

El tercer inconveniente surgido es reuso de conceptos. Para que la facilidad de utilizar varias presentaciones con un solo concepto pueda ser implementada es necesario

construir herramientas que permitan explorar el grafo en busca de conceptos reusables, crear las nuevas presentaciones y asociar estos conceptos con las nuevas presentaciones.

Diferencias con el trabajo de Oinas

Mientras estabamos preparando este trabajo de grado pudimos leer un trabajo de Harry Oinas que mostraba los beneficios que se podían obtener anexando una interfaz hipermedial a una herramienta CASE.

Nos aprecio útil describir las diferenciar nuestro trabajo con el de Oinas.

Las ópticas son sutilmente diferentes, la óptica de Oinas consiste en anexar una interfaz hipermedial a una herramienta CASE pensada para funcionar de otra manera, nosotros diseñamos una herramienta CASE para funcionar con interfaz hipermedial aunque pueda disponer de otro tipo de herramientas.

Granularidad

Para el mínimo nivel de nodo es el de documento.

Para nosotros cada figura del documento puede ser un nodo que tiene distintas formas de ser percibido

Links:

Para el son solo elementos de navegación

Para nosotros son representaciones perceptibles de relaciones, las cuales pueden ser usadas para chequear consistencia, etc.

Generación de los links:

Para el deben ser generados explícitamente por el usuario

Nosotros creemos que hay links (Asociaciones) que pueden ser creadas automáticamente.

Editores gráficos:

No creemos que sea buena de tener una única representación gráfica que soporte varios tipos de documentos, las notaciones son importantes en la aplicación de las metodologías y deben proveerse editores que reflejen las características visuales de cada documento.

Links y consistencia

Según Harry la interfaz hipermedial y el chequeo de consistencia son independientes,

En nuestro caso la integración es un tema de mucha importancia y determina la dependencia entre estas características utilizar la misma información para navegar y chequear consistencia.

Con este esquema aparece un problema con el que Harry no tiene que lidiar tenemos un nodo con una gran cantidad de links para presentar lo cual es muy complicado para el usuario entonces hay que implementar esquemas de filtros para que el usuario vea solo lo que necesita y no le estorbe el resto, hay que generar una función

del documento, el usuario el browser y el estado del case para presentar al usuario un conjunto MVC adecuado.

Conclusiones

Como resultado de nuestro trabajo obtuvimos una herramienta de ingeniería de software, desarrollada con tecnologia de objetos, basada en un repositorio en forma de grafo, con interfaz hipermedial que:

Alcanza los objetivos de *extensibilidad* y *modificacbilidad* propuestos inicialmente.

Lo pudimos comprobar cuando se construyó el prototipo. Pudimos hacer una construcción incremental desde el repositorio hacia las herramientas y comprobamos que el desarrollo de nuevos componentes no tenían impacto significativo en las componentes ya construidas.

Del análisis hecho de las posibles extensiones que se le pueden hacer a HCase surge una muy buena expectativa de que la incorporación de nueva funcionalidad mantenga el bajo coeficiente de impacto.

Esta provista de herramientas que permiten personalizar muchas características importantes.

Posee *editores gráficos* potentes y expresivos.

Combina la expresividad de los editores gráficos con la formalidad del *lenguaje definida por la metodología.*

Maneja en forma *consistente* la gran cantidad y complejidad de la documentación generada en los distintos desarrollos y las dependencias entre los distintos documentos.

Provee al usuario distintos niveles de abstracción.

Incorpora conocimientos específicos y reglas de manera de *ayudar al usuario a aplicar la metodología.*

Mantiene *multiples proyectos* simultaneamente.

Explota la potencia de la *interfaz hipermedial* en la búsqueda, acceso y reuso de los documentos generados en los desarrollos.

Brinda soporte para *formas poco estructuradas de relacionar información.*

Con algunas de las extensiones previstas es capaz de dar soporte para *el ciclo de vida completo.*

Esta *integrada* dentro del ambiente en el que se hacen los desarrollos, evitando el uso de diferentes ambientes para la documentación y el desarrollo, y utilizando las *descripciones y herramientas ya existentes.*

Es *simple de aprender y utilizar*, desarrollar con HCase es sencillamente crear, explorar, modificar o eliminar documentos y asociaciones entre documentos.

Incorpora ideas novedosas como la generación automática y dinámica de links o la visión del link como representante de una asociación, esto ultimo deriva en la utilización de links para el testeo de consistencia.

Propone estrategias de reuso.

Queremos agradecer a nuestros directores que tuvieron gran incidencia en algunas decisiones acertadas, determinantes para el éxito del trabajo.

La utilización de Smalltalk, porque es una excelente herramienta de modelización.

La utilización de un repositorio.

La modelación del repositorio en forma de grafo.

La extensión de frameworks como técnica de construcción.

El hallazgo del FrameWork HotDraw que nos permitió desarrollar una interfaz muy potente.

La elección de modelar las asociaciones entre los distintos documentos como objetos individuales y no como atributos de los documentos

Consideramos que el trabajo realizado tuvo dos desafíos importantes,

Por un lado lograr la integración de distintos dominios de aplicación como Objetos, Hipermedia , CASE, Metodologías de desarrollo y Repositorios requirió un fuerte trabajo de investigación previa, de manera de lograr un conocimiento lo suficientemente profundo de cada uno de ellos y obtener un punto de vista integrador.

Por otro lado la construcción del prototipo genero la necesidad de utilizar diferentes herramientas, para esta tarea debimos aprender Lenguajes de programación (Smalltalk), Frameworks (HotDraw, MVC) y metodología de desarrollo.

En ambos casos la complejidad de la tarea fue, fundamentalmente, la integración de diferentes conceptos y tecnologías.

Creemos que gran parte de la complejidad del tema CASE no solo pasa por la gran cantidad de requerimientos que presenta. La gran cantidad de herramientas, de distintos tipos, y la gran integración que se pretende de ellas es una de las causas fundamentales para que el análisis y desarrollo de aplicaciones, en este dominio, sea de gran complejidad.

La definición de los alcances y el diseño preliminar fueron una de las partes mas difíciles del trabajo. El dominio de aplicación de las herramientas CASE es muy amplio. Una herramienta CASE, combina muchas tecnologías, domina muchos conceptos, provee mucha funcionalidad, todo esto en forma integrada. Debíamos mostrar que nuestra herramienta era adaptable y extensible pero primero debíamos definir los alcances de la hrramienta de manera de mostrar que era capaz de alcanzar los requerimientos necesarios.

Tanta amplitud provocó que necesitemos un período de divergencia muy largo antes de converger hacia un resultado.

Como corolario cabe decir que estamos orgullosos de los resultados obtenidos, y profundamente agradecidos, porque los conocimientos adquiridos nos son de gran utilidad en el mercado laboral en el que estamos insertos.

Trabajo futuro

Prueba de modificabilidad y adaptabilidad

Si bien la construcción del prototipo ya nos dio una buena idea del nivel de modificabilidad y adaptabilidad de HCase creemos importante hacer una experiencia exhaustiva, construyendo el conjunto de extensiones propuestas y obteniendo mediciones concretas.

Prueba con usuarios

Otra de las tareas que desearíamos hacer es un estudio de la reacción de los usuarios frente a HCase, la experiencia consistiría en entregar la herramienta a un conjunto de "Testers" y evaluar las opiniones.

Extensiones complejas

Durante el desarrollo de HCase quedaron pendientes algunos dominios de aplicación que también tienen que ver con herramientas de ingeniería de software, quedo como objetivo pendiente profundizar sobre estos temas.

Análisis del impacto de las modificaciones

Una de las ideas que surgieron al relevar los requerimientos de las herramientas CASE es la de diseñar una herramienta que evalúe el impacto que puede producir la modificación de un elemento que ya ha sido desarrollado.

Para esto deseamos aprovechar la posibilidad que presenta HCase al permitir presentar el repositorio desde distintos puntos de vista, en particular el punto de vista del grafo y el de un dominio al que se le pueden aplicar predicados. Utilizando los puntos de vista alternativos deseamos definir un conjunto de reglas que mida el impacto y herramientas que implementen estas reglas.

Era obvio que el impacto dependería del tipo y la cantidad de nodos que quisiéramos modificar, de la cantidad y tipo de arcos que llegaran a ellos y de como esos arcos conectaban nodos críticos o periféricos del grafo. Creemos que una facilidad de este tipo sería posible.

Bases de datos

Sería importante anexar a las funcionalidades de HCase facilidades de persistencia para los desarrollos generados.

Versiones

También es importante proveer al desarrollador la posibilidad de versionar los desarrollos y medios para mantener las distintas versiones consistentes y actualizables.

Trabajo en grupos

Existen otras características que son útiles para desarrollos importantes realizados por grupos. Algunos de ellos son:

- La posibilidad de que varios desarrolladores trabajen concurrentemente sobre los mismos objetos, manteniendo la consistencia.
- Permitir distintas formas de percibir la información según el rol que desempeña cada uno en el desarrollo.
- Proveer seguridad y niveles de acceso a la información generada.

Bibliografía

[Ber'96] Bernstein P.(1996) The repository, a modern Vision. DBPD, Septiembre'96.

[Bieber y Kimbrough] Bieber MP &Kimbrough SO (1992) On generalizing the concept of hipertext. MIS Quarterly 16(1)

[Booch'94] Booch G. (1994)Object-Oriented Analysis and Design with Applications , Benjamin/Cummings, Redwood City, Calif.

[Burton'87] Burton B. Et al. (1987), The Reusable Software Library, IEEE Software, July 1987.

[Coad'91] Coad P. Y Yourdon E. (1991). Object-Oriented Analisys 2nd edtn. Eglewood Cliffs, NJ: Prentice Hall.

[Conklin] Conklin J. (1987) Hipertext: an introduction and survey. IEEE Computer 20(9):

[Cybulsky y Reed] Cybulsky JL & Reed K (1992) A Hipertext-based software engineering environment. IEEE Software 9(2)

[Deutsch] L. Peter Deutsch, Design Reuse and Framework in the Smalltalk-80 Programming System, página 55-71, software Reusability, Vol II de. Ted J. Biggerstaff and Alan J. Perlis, ACM Press. 1989.

[Fuggeta) Fuggeta A (1993) A clasification of CASE technology. IEEE Computer 26(12)

[Garg'87] Garg PK & Scacchi W (1987) On designing Intelligent Hypertext Systems for Information Management in Software Engineering. Proceedings of Hypertext' 87.

[Garg'89] Garg PK (1989) Ishys. Designing an Intelligent Software Hypertext Sysytem. Fall 1987.

[Garg'90] Garg PK & Scacchi W (1990) A hypertext system to manage software life-cycle documents. IEEE Software 7(3)

[Jac'92] Jacobson I., Christerson M., Jonsson P. And Overgaaard G. (1992). Oject-Oriented Software Engeneering - A Use Case Driven Approach. Addison-Wesley.

[Jac'94] Jacobson I., Ericsson Maria, Jacobson Agneta (1994) The Object Advantage: Bussines process reengineering with object technology.

[Jac'96] Jacobson I. (1996), Succeding With Objects: Reuse in Reality, Object Magazine, Julio 1996.

[Johnson] Ralph E. Johnson Documenting Frameworks using Patterns. Department of Computer Sciencel 1304 W. Springfield Ave. Urbana, IL, 61801

[Johnson y Footer] Ralph E. Johnson and Stephen T. Pope (1988), A Cookbook for Using the Model-View-Controller User Interface Paradigin in Smalltalk-80, Journal of Object Oriented Programmig 1(3).

[Lalonde'91a] Wirf R. Lalonde M. y Pugh J.(1991) Inside Smalltalk. Volume II. Eglewood Cliffs, NJ: Prentice Hall.

[Lalonde'91b] LaLonde W., Discovering Smalltalk (1991), Eglewood Cliffs, NJ: Prentice Hall.

[Nielsen'90] Nielsen J. Hypertext and Hypermedia. Academic Press, 1990.

[OMG'94] Object Management Group (1994). Object Analysis and Design. Description of methods. (Hutt A.T.F., de.). New york: John Wiley & Sons

[Rum'91] Rumbaugh J., Blaha M., Premerlani W., Eddy F., y Lorensen W. (1991) Object-Oriented Modeling and Design. Englewood Cliffs, NJ: Prentice Hall.

[Oinas'93a] Oinas-Kukkonen H. (1993) Hypertext Functionality in CASE Environments: Preliminary Findings. Proceedings de Hypermedia en Vaasa'93.

[Oinas'93b] Oinas-Kukkonen H. (1993) Hypertext in CASE Environments: Working definitions and Taxonomies. Research papers Series A16. University of Oulu.

[Wirf'90] Wirf-Brock R., Wilkerson B. And Wienter L. (1990). Designing Object-Oriented Software. Englewood Cliff.

Diseño de las clases básicas de HCase

INDICE

Introducción

A continuación se lista la descripción y codificación de las principales clase del proyecto. La clases que definen la interfáz con el usuario no son detalladas dado que la mayoría están realizadas con el Canvas provisto por Smalltalk y no tienen detalle del diseño.

HDocumento

```
Object ()
        Model ('dependents')

                HDocumento ('nombre' 'comentario' 'autor' 'fechaDeCreacion' 'keyWord'
'vistas' 'fechaDeUltimaModificacion')
                        ArcoDeRepositorio ('tipo' 'origen' 'destino' 'label')
                                ArcoConcreto ()
                                ArcoVirtual ()
                        NodoDeRepositorio ('asociaciones')
                                ConceptoDeJacobson ()
                                        Actor ()
                                        Atributo ()
                                        CasoDeUso ()
                                        Especificacion ('especificacion')
                                        ModeloDeAnalisis ()
                                        ModeloDeCasosDeUso ()
                                        ModeloDeDiseno ()
                                        ObjetoControl ()
                                        ObjetoEntidad ()
                                        ObjetoInterfaz ()
                                        Proyecto ('dominiosDeAplicacion')
                                                ProyectoDeJacobson ()
                                        RepositorioDeProyectos ()
                                        Subsistema ()
                                        TarjetaDeClase ('responsabilidades' 'superclase'
'subclases' 'abstracta')
```

Class: **HDocumento**

Superclass: Model
Category: HCase-Framework
Instance variables: nombre comentario autor fechaDeCreacion keyWord vistas fechaDeUltimaModificacion

Un HDocumento es una clase abstracta que representa cualquier cosa que pueda aparecer en el repositorio, contiene los descriptores basicos que poseen todos los items, el comportamiento basico referente a las vistas (dado que cualquier item en el repositorio puede ser presentado al usuario y posee todo el comportamiento comun a cualquier objeto que pueda estar en el repositorio,

Las variables de instancia son:

nombre - Symbol. Cada Item tendra asociado con un nombre para su identificacion.
comentario - Text. Una descripcion del item.
autor - Symbol. Nombre del autor/es.
fechaDeCreacion - Date. Fecha en la que fue creado el item
keyWord - Symbol. Palabra que permite al usuario clasificar los items para realizar busquedas positeriores.
vistas - Collection. Mantiene una coleccion con las vistas dependientes.

fechaDeUltimaModificacion - Date. Fecha en la que se introdujo la ultima modificacion en el item

private

modificado
"Cada vez que se modifica el receptor se actualiza la fecha de ultima modificacion"

fechaDeUltimaModificacion := Date today

accessing

autor
"Retorna el nombre del autor del receptor"

^autor

autor: unSymbol
"Asigna el nombre del autor del receptor a unSymbol"

autor := unSymbol.
self changed.
"self actualizarVistas: #CambioAutor"

comentario
"Retorna el comentario del item"

^comentario

comentario: unText
"Asigna el comentario del receptor a unText"

comentario := unText asText.
self changed.
self actualizarVistas: #CambioComentario

fechaDeCreacion
"Retorna la fecha de creacion del reseptor"

^fechaDeCreacion

fechaDeCreacion: unaFecha
"Asigna la fecha de creacion del receptor a unaFecha"

fechaDeCreacion := unaFecha.
self actualizarVistas: #CambioFechaDeCreacion

fechaDeUltimaModificacion
"Retorna la fecha de ultima modificacion del receptor"

^fechaDeUltimaModificacion

fechaDeUltimaModificacion: unaFecha
"Asigna la fecha de ultima modificacion del receptor a unaFecha"

fechaDeUltimaModificacion := unaFecha.
self actualizarVistas: #CambioFechaDeUltimaModificacion

```
keyWord
        "Retorna el keyWord del receptor"

        ^keyWord

keyWord: unString
        "Asigna la keyWord del receptor a unString"

        keyWord := unString.
        self changed.
        self actualizarVistas: #CambioKeyWord

nombre
        "Retorna el nombre del receptor"

        ^nombre

nombre: unSymbol
        "Asigna el nombre del receptor a unSymbol"

        nombre := unSymbol.
        self changed.
        self actualizarVistas: #CambioNombre
```

case properties

```
tipoDeDocumento
        "Retorna el tipo del receptor. Es el nombre de la clase"

        ^self class name
```

initialize-release

```
initialize
        "Inicializa el receptor con los descriptores por defecto"

        nombre := #UnItem.
        comentario := Text new.
        autor := #Anonimo.
        fechaDeCreacion isNil ifTrue: [fechaDeUltimaModificacion := fechaDeCreacion :=
Date today].
        keyWord := #Normal.
        vistas := Set new.
        ^self

initializeNombre: unSymbol
        "Inicializa el item con los descriptores por defecto, excepto el nombre que es
inicializado con
        unSymbol"

        self initialize.
        nombre := unSymbol.
        ^self
```

vistas

```
actualizarVistas: unSymbol
        "Actualiza las vistas dependientes del receptor de acuerdo con unSymbol"
```

```
        self actualizarVistas: unSymbol con: nil

actualizarVistas: unSymbol con: unParametro
        "Actualiza las vistas dependientes del receptor de acuerdo con unSymbol y
        unParametro"

        self
                actualizarVistas: unSymbol
                con: unParametro
                y: nil

actualizarVistas: unSymbol con: unParametro y: otroParametro
        "Actualiza las vistas dependientes del receptor de acuerdo con unSymbol,
        unParametro y otroParametro"

        vistas do: [:vis | vis
                        actualizar: unSymbol
                        con: unParametro
                        y: otroParametro]

agregarVista: unaVista
        "Agrega unaVista a la lista de vistas del receptor"

        vistas add: unaVista

eliminarVista: unaVista
        "Elimina unaVista de la lista de vistas del receptor"

        vistas remove: unaVista

view
        "Muestra la ventana por defecto del Item"

        self actualizarVistas: #Mostrarse con: #Tarjeta

vistas
        "Retorna las vistas instanciadas del receptor"

        ^vistas
```

changing

```
changed
        "Alguna de las propiedades del receptor han cambiado"

        self changed: #changedDescriptor
```

removing

```
remove
        "Remueve definitivamente el receptor del repositorio. Informa a sus dependientes que
        él será removido"

        self actualizarVistas: #remove.
        self changed
```

MetaClass: ***HDocumento class***

instance creation

```
new
        "Crea una nueva instancia del receptor"

        ^super new initialize

newNombre: unSymbol
        "Crea una nueva instancia de receptor cuyo nombre es unSymbol"

        super new initializeNombre: unSymbol
```

vistas

```
vistaPorDefecto
        "Retorna la clase de la vista por defecto del receptor"

        ^TarjetaDeItemDeRepositorio
```

Class: ArcoDeRepositorio

Superclass: HDocumento
Category: HCase-Rep-Support
Instance variables: tipo origen destino label

Los arcosDeRepositorio son los arcos del grafo, cuando se modeliza al repositorio de esta manera, contienen la informacion de las asociaciones entre los distintos documentos u objetos de la metodologia, son la base para la navegacion (Links de hipermedia), los modelos para las flechas en los editores graficos, etc.

Las variables de instancia son:

tipo Symbol - Representa el tipo de la asociacion.
origen NodoDeRepositorio.
destino NodoDeRepositorio.
label String. Representa el label que tendra el arco cuando sea presentado en un editor grafico

accessing

```
destino
        "Retorna el nodoDeRepositorio el cual es destino del arco"

        ^destino

destino: unItemDeRepositorio
        "Asigna el destino del arco con unItemDeRepositorio"

        destino := unItemDeRepositorio.
        self actualizarVistas: #CambioDestino

label
        "Retorna el label del receptor"
```

```
        ^label

label: unString
        "Asigna el label y el nombre del arco a unString"

        label := unString.
        nombre := unString.
        self actualizarVistas: #CambioLabel

origen
        "Retorna el nodoDeRepositorio el cual es origen del receptor"

        ^origen

origen: unItemDeRepositorio
        "Asigna el origen del arco a unItemDeRepositorio"

        origen := unItemDeRepositorio.
        self actualizarVistas: #CambioOrigen

tipo
        "Retorna el tipo del receptor"

        ^tipo

tipo: unSymbol
        "Asigna el tipo del arco a unSymbol"

        tipo := unSymbol.
        self actualizarVistas: #CambioTipo
```

initialize-release

```
initialize
        "Inicializa el item con los descriptores por defecto"

        super initialize.
        label := String new.
        tipo := #SinTipo.
        ^self
```

case properties

```
inverza
        "Retorna un arco virtual con el destino y el origen invertidos, y el tipo como el tipo
inverza del receptor"

        ^(ArcoVirtual new) origen: self destino; destino: self origen; tipo: (self tipoInverzaDe:
self tipo)

orientarseDesde: unNodoOrigen
        "Retorna un arco con el origen en unNodoOrigen y tipo adecuado"

        self origen == unNodoOrigen ifTrue: [^self].
        self destino == unNodoOrigen ifTrue: [^self inverza].
        ^nil

tipoInverzaDe: unSymbol
        "Retorna el tipo inverza de unSymbol"
```

```
        | newSymbol |
        unSymbol == #ParteDe ifTrue: [^#ContieneA].
        unSymbol == #Extends ifTrue: [^#Extended].
        unSymbol == #Aquaintance ifTrue: [^#Aquaintance].
        unSymbol == #iniciaA ifTrue: [^#IniciadoPor].
        unSymbol == #Uses ifTrue: [^#Used].
        newSymbol := Symbol new: 'inverzaDe' size + unSymbol size.
        ^(newSymbol
                copyReplaceFrom: 1
                to: 'inverzaDe' size
                with: 'inverzaDe')
                copyReplaceFrom: 'inverzaDe' size + 1
                to: newSymbol size
                with: unSymbol
```

testing

```
conecta: unConcepto
        "Retorna true si unConcepto es alguno de los extremos del arco"

        ^self destino = unConcepto or: [self origen = unConcepto]

entre: unConcepto y: otroConcepto
        "Retorna true si unConcepto y otroConcepto son los extremos del arco"

        ^(destino = unConcepto and: origen = otroConcepto)
                or: (destino = unConcepto and: origen = otroConcepto)
```

converting

```
asSymbol
        "Retorna newSymbol el describe las componentes del receptor"

        | simboloDelTipo simboloDelDestino newSymbol simboloDelOrigen |
        simboloDelTipo := self tipo asSymbol.
        simboloDelDestino := self destino nombre asSymbol.
        simboloDelOrigen := self origen nombre asSymbol.
        newSymbol := Symbol new: simboloDelTipo size + simboloDelDestino size +
simboloDelOrigen size + 2.
        newSymbol := newSymbol
                                copyReplaceFrom: 1
                                to: simboloDelOrigen size
                                with: simboloDelOrigen.
        newSymbol := newSymbol
                                copyReplaceFrom: simboloDelOrigen size + 1
                                to: simboloDelOrigen size + 1
                                with: ' '.
        newSymbol := newSymbol
                                copyReplaceFrom: simboloDelOrigen size + 2
                                to: simboloDelOrigen size + 2 + simboloDelTipo size
                                with: simboloDelTipo.
        newSymbol := newSymbol
                                copyReplaceFrom: simboloDelOrigen size + 2 +
simboloDelTipo size
                                to: simboloDelOrigen size + 2 + simboloDelTipo size
                                with: ' '.
        newSymbol := newSymbol
```

```
                                copyReplaceFrom: simboloDelOrigen size + 2 +
simboloDelTipo size + 1
                                to: newSymbol size
                                with: simboloDelDestino.
        ^newSymbol asSymbol

asSymbolOrigen: unNodoOrigen
        "Retorna newSymbol el describe las componentes del receptor cuyo nodo origen es
        unNodoOrigen"

        | simboloDelTipo simboloDelDestino newSymbol arco |
        arco := self orientarseDesde: unNodoOrigen.
        simboloDelTipo := arco tipo asSymbol.
        simboloDelDestino := arco destino nombre asSymbol.
        newSymbol := Symbol new: simboloDelTipo size + simboloDelDestino size + 1.
        newSymbol := newSymbol
                                copyReplaceFrom: 1
                                to: simboloDelTipo size
                                with: simboloDelTipo.
        newSymbol := newSymbol
                                copyReplaceFrom: simboloDelTipo size + 1
                                to: simboloDelTipo size + 1
                                with: ' '.
        newSymbol := newSymbol
                                copyReplaceFrom: simboloDelTipo size + 2
                                to: newSymbol size
                                with: simboloDelDestino.
        ^newSymbol asSymbol
```

MetaClass: *ArcoDeRepositorio class*

vistas

```
vistaPorDefecto
        ^TarjetaDeArcoDeRepositorio
```

Class: NodoDeRepositorio

Superclass: HDocumento
Category: HCase-Rep-Support
Instance variables: asociaciones

NodoDeRepositorio es una calse abstracta que encapsula las responsabilidades comunes de los nodo del repositrio. Esta clase cuenta con un conjunto estandares de responsabilidades que están orientada la construcción de query.

Variables de instancias:

asociaciones -Set. Cada componente del Set es una instancia de una subclase de ArcoDeRepositorio

initialize-release

```
initialize
        "Inicializa el receptor"
```

```
        super initialize.
        asociaciones := Set new.
        ^self
```

asociaciones

```
asociacionContenedor
        "Retrona el primer arco en la cual el receptor es #parteDe del origen"

        ^(self asociacionesDirectasTipo: #ParteDe) first

asociaciones
        "Retorna el Set de asociaciones del receptor"

        ^asociaciones

asociacionesCon: unConcepto tipo: unSymbol label: unString
        "Retorna una coleccion ordenada con los arcos que satisfacen que el origen o el
        destino sea unConcepto y el tipo unSymbol. Evalua el parametro solo si no es nil"

        ^(self asociaciones select: [:t1 | (((t1 origen = unConcepto or: unConcepto isNil)
                        or: t1 destino = unConcepto)
                        and: (t1 tipo = unSymbol or: unSymbol isNil))
                        and: (t1 label = unString or: unString isNil)]) asOrderedCollection

asociacionesDirectas
        "Retona una colleción ordenada con todo los arcos del receptor asociados de forma
directa"

        ^self
                asociacionesDirectasCon: nil
                tipo: nil
                label: nil

asociacionesDirectasCon: unConcepto tipo: unSymbol label: unString
        "Retorna una coleccion ordenada con los arcos que satisfacen que el destino es
unConcepto, el
        tipo unSymbol y el origen es el receptor. Evalua el parametro solo si no es nil"

        ^(self asociaciones select: [:t1 | (((t1 destino = unConcepto or: unConcepto isNil)
                        and: (t1 tipo = unSymbol or: unSymbol isNil))
                        and: (t1 label = unString or: unString isNil))
                        and: t1 origen = self]) asOrderedCollection

asociacionesDirectasTipo: unSymbol
        "Retorna los arcos cuyo orgen es el recptor y el tipo del arco es igual a unSymbol"

        ^self
                asociacionesDirectasCon: nil
                tipo: unSymbol
                label: nil

asociacionesInverza
        "Retorna toda los arcos donde el receptor es destino del arco"

        ^self
                asociacionesInverzaCon: nil
                tipo: nil
                label: nil
```

```
asociacionesInverzaCon: unConcepto tipo: unSymbol label: unString
        "Retorna una coleccion ordenada con los arcos que satisfacen los valores de
        los parámetros. Evalua el parametro solo si no es nil. En este caso toma la inversa de
        la relación"

        | unSymbolInversa |
        unSymbolInversa := self inversaDe: unSymbol.
        ^(self asociaciones select: [:t1 | (((t1 origen = unConcepto or: unConcepto isNil)
                        and: (t1 tipo = unSymbolInversa or: unSymbolInversa isNil))
                        and: (t1 label = unString or: unString isNil))
                        and: t1 destino = self]) asOrderedCollection

asociacionesInverzaTipo: unSymbol
        "Retorna todo los arcos en donde el receptor es destino del mismo y el tipo es
        unSymbol"

        ^self
                asociacionesInverzaCon: nil
                tipo: unSymbol
                label: nil

asociacionesOrigen: unConcepto destino: otroConcepto tipo: unSymbol label: unString
        "Retorna una coleccion ordenada con los arcos que satisfacen que el origen es
        unConcepto, el destino otroConcepto y el tipo unSymbol. Evalua el parametro solo si
        no es nil"

        ^(self asociaciones select: [:t1 | (((t1 origen = unConcepto or: unConcepto isNil)
                        and: (t1 destino = otroConcepto or: otroConcepto isNil))
                        and: (t1 tipo = unSymbol or: unSymbol isNil))
                        and: (t1 label = unString or: unString isNil)]) asOrderedCollection

asociacionesTipo: unSymbol
        "Retorna los arcos en donde el receptor es origen y el tipo es unSymbol"

        ^self
                asociacionesCon: nil
                tipo: unSymbol
                label: nil

asociacionesTipo: unSymbol con: unConcepto
        "Retorna los arcos en donde el origen es el receptor, el destino unConcepto y el tipo
        unSymbol"

        ^self
                asociacionesCon: unConcepto
                tipo: unSymbol
                label: nil

asociadosDirectos
        "Retorna una colección con los conceptos destino asociados directamente al receptor"

        ^self asociacionesDirectas collect: [:aso | aso destino]

asociadosInversos
        "Retorna una colleción con los conceptos origenes asociados al receptor"

        ^self asociacionesInverza collect: [:aso | aso origen]

contenedor
        "Retorna el primer concepto que contine al receptor"
```

```
        ^self asociacionContenedor destino

diccionarioDeAsociados
        "Retorna un diccionario donde la clave es la denominación del origen y la componente
        el correspondiente destino"

        | diccio |
        diccio := Dictionary new.
        self asociaciones do: [:aso | diccio at: (aso asSymbolOrigen: self)
                                put: (aso orientarseDesde: self) destino].
        ^diccio

tiposDeAsociaciones
        "Retrona un Set con todo los tipos de arcos asociados al receptor"

        ^(self asociaciones collect: [:arco | arco tipo]) asSet

tiposDeAsociacionesDirectas
        "Retorna un set con los tipos de los arcos asociados al receptor de forma directa"

        ^(self asociacionesDirectas collect: [:arco | arco tipo]) asSet

tiposDeAsociacionesInverza
        "Retorna un Set con los tipos de los arcos asociados de forma inversa al receptor"

        ^(self asociacionesInverza collect: [:aso | aso tipo]) asSet

traceabiltyDirecta
        "Retorna una colección de arcos cuyo destino está asociado por #Traceabilty con el
        receptor"

        ^self asociacionesDirectasTipo: #Traceability

traceabiltyInversa
        "Retorna una colección de arcos en donde el origen esta asociado por #Traceability
        con el receptor"

        ^self asociacionesInversaTipo: #Traceability

private

linkAdd: unaArco
        "Agrega un arco a las lista de asociaciones del receptor"

        asociaciones add: unaArco

linkRemove: unArco
        "Remueve unArco de la lista de asociaciones del receptor"

        asociaciones remove: unArco

removeAsociaciones
        "Remueve definitivamente los conceptos asociados al receptor del repositorio"

        asociaciones do: [:aso | aso remove]
```

case properties

```
dominiosDeAplicacion
        "Retorna el dominio de aplicación al cual pertenece el proyecto corriente"
```

```
        ^self proyecto dominiosDeAplicacion

etapa
        "Retorna la etapa del ciclo de vida la cual modela el receptor"

        ^'Desarrollo'

proyecto
        "Retorna el proyecto que contiene al receptor"

        ^self contenedor proyecto
```

removing

```
remove
        "Remueve el receptor del repositorio de forma definitiva"

        super remove.
        self removeAsociaciones
```

MetaClass: *NodoDeRepositorio class*

vistas

```
vistaPorDefecto
        "Retorna la clase de la vista por defecto del receptor"

        ^TarjetaDeNodoDeRepositorio
```

Class: ConceptoDeJacobson

Superclass: NodoDeRepositorio
Category: HCase-Rep-Support

ConceptoDeJacobson es un clase abtracta que especializa ciertos aspectos particulares de los conceptos de la metodología de Jacobson.

partes

```
partes
        "Retorna una colección con los conceptos que son parte del
        receptor"

        | partesEncontradas |
        partesEncontradas := self asociacionesInverzaTipo: #ParteDe.
        ^partesEncontradas collect: [:parte | parte origen]

partesAdd: unNodo
        "Agrega un arco al repositorio en donde el destino es el receptor, el origen es unNodo
        y el tipo de relación es #ParteDe"
```

```
            ^ArcoConcreto
                destino: self
                origen: unNodo
                tipo: #ParteDe
                label: 'Parte de'

    partesNombre
            "Retorna una colección con los nombre de toda las componentes del receptor"

            ^self partes collect: [:each | each nombre asString]

    partesNombre: unSymbol
            "Retorna una colección con las componentes del receptor cuyo nombre
            es unSymbol"

            | partesEncontradas |
            partesEncontradas := self partes.
            ^partesEncontradas detect: [:par | par nombre = unSymbol]

    partesTipo: unSymbol
            "Retorna una colección con las componentes del receptor cuyo tipo de relación
            es unSymbol"

            ^self partes collect: [:par | par tipo = unSymbol]

    remove: unNodo
            "Remueve unNodo el cual es una componete del receptor. Si el nodo no es parte o np
            existe no hace nada"

            (self
                asociacionesInverzaCon: unNodo
                tipo: #ParteDe
                label: nil) first remove
```

case properties

```
    ultimaModificacion
    "La ultima modificación de un ConceptoDeJacobso se define como la ultima fecha de
modificación de sus partes"
            | par |
            par := self partes.
            ^par inject: par first ultimaModificacion into: [:res :parte | res max: parte
ultimaModificacion]
```

testing

```
    estasVacio
            "Retorna true si el receptor no tiene partes, si no false"

            ^self partes isEmpty

    includes: unNodo
            "Retorna true si unNodo es parte del receptor, si no false"

            ^self partes includes: unNodo
```

HEditor

```
Object ()
        Model ('dependents')
                DrawingEditor ('tools' 'currentTool' 'fileName' 'drawing')

                        HEditor ('navegador')
```

Class: HEditor

Superclass: DrawingEditor
Category: HCase-Framework
Instance variables: navegador

HEditor son DrawingEditor especializados para adquirir servicios de hypermedia. Cada nodo del repositorio es presentado al espectador incrustando una HPresentacionCompuesta (subclase de Drawing) a un HEditor, cada nodo es apilado en el navegador y el contenido del mismo se permite editar. Cada componete de HEditor es un nodo del hyperespacio y a su vez un diagrama el cual puede ser editado por el espectador.

Un HEditor interactua con una única instacia global del repositorio que tenemos en el prototipo, la cual se llama HCase esta es instancia de RepositorioDeProyectos. Instancia de HEditor juegan el papel de browser hypermedial del repositorio.

Un HEditor tambien permite personalizar o extender conceptos de la metodología de turno, por lo cual tambien se tiene acceso a una instancia global de HMetodologia que se llama MetodologiaOOSE por medi de un editor apropiedo.

Variables de instancias:

navegador HNavigationManager. Manejador para navegar el hyperespacio.

accessing

```
drawing
        "Retorna el corriente drawing (HPresentaciónCompuesta)"

        ^drawing

menu
        "Retorna el menú comun a todo los drawing"

        ^PopUpMenu
                labels: 'adelante\atras\navegación\propiedades\asociaciones\reglas' withCRs
                lines: #(2 5 )
                values: #(#adelante #atras #navegacion #propiedades #navegarAsociacion
#reglas )

metodologia
        "Inicialmente tenemos una única metodología. Este concepto se puede extender a varias
        metologías y se define una inicialización para determinar la corriente. Subclases de
        HMetodologia permiten definir metodologías que combinara HCase."

        ^MetodologiaOOSE

metodologiaTool
        "Abre la herramienta que permite personalizar las definiciones de la Metodología
        corriente. Solo puede haber una metodología activa por vez"
```

```
        ^HMetodologiaTool openOn: self metodologia

navegador
        "Retorna el navegador del receptor"

        ^navegador

navegador: aNavigationManager
        "Asigna al navegador del receptor con aNavgationManager"

        navegador := aNavigationManager
```

private

```
cambiarNodo: unNodo
        "Hace que el receptor muetre unNodo como el corriente drawing"

        | aLabel |
        self currentTool controller view noSelections.
        self setDrawing: unNodo.
        aLabel := self armarLabel: unNodo.
        HBrowser label: aLabel

refrescar
        "Refresca el aspecto visual del corriente drawing del receptor con sus
        correspondientes dependientes (Model)"

        | myToolsViews myDrawingViews |
        myToolsViews := self myDependents select: [:each | each class == ToolPaletteView].
        myToolsViews
                do:
                        [:each |
                        each invalidate.
                        each repairDamage].
        myDrawingViews := self myDependents select: [:each | each class == DrawingView].
        myDrawingViews do: [:each | each setDrawing: self drawing].
        myDrawingViews
                do:
                        [:each |
                        each invalidate.
                        each repairDamage].
        self tools do: [:each | each controller: myDrawingViews first controller]
```

utilidades

```
armarLabel: unNodo
        "Arma el titulo de la ventana del reseptor. Por lo general la ventana de HCase"

        ^unNodo propietario class name asString , '-HCase System'
```

hypermedia

```
abrir
        "Abre un browser para el reseptor donde el primer nodo es
        la raiz"
```

```
        | aDrawingView aToolPaletteView aWindow container toolPaletteSize aDrawingEditor
adelante atras buttonsNavegar buttonsHyper navegacion nodo aLabel primerNodo metodologia |
        self breakDependents.
        aDrawingEditor := self.
        primerNodo := self navegador nodoAt: 1.
        self setDrawing: primerNodo.
        self navegador clearPila.
        self navegador pushNodo: primerNodo.
        aDrawingView := DrawingView on: aDrawingEditor.
        aDrawingView setDrawing: primerNodo.
        aToolPaletteView := ToolPaletteView on: aDrawingEditor.

        buttonsNavegar := CompositePart new.
        adelante := (Button trigger) model: ((PluggableAdaptor on: aDrawingEditor)
                                        performAction: #adelante); label: '>>'.
        atras := (Button trigger) model: ((PluggableAdaptor on: aDrawingEditor)
                                        performAction: #atras); label: '<<'.
        buttonsNavegar add: atras in: (0 @ 0 corner: 0.5 @ 1).
        buttonsNavegar add: adelante in: (0.5 @ 0 corner: 1 @ 1).
        buttonsHyper := CompositePart new.
        metodologia := (Button trigger) model: ((PluggableAdaptor on: aDrawingEditor)
                                        performAction: #metodologiaTool); label:
'Metodología'.
        navegacion := (Button trigger) model: ((PluggableAdaptor on: aDrawingEditor)
                                        performAction: #navegacion); label: 'Navegación'.
        buttonsHyper add: navegacion in: (0 @ 0 corner: 0.5 @ 1).
        buttonsHyper add: metodologia in: (0.5 @ 0 corner: 1 @ 1).
        nodo := CompositePart new.
        toolPaletteSize := 280.
        nodo add: aToolPaletteView borderedIn: (0 @ 0 extent: 4 + Tool toolImageSize x +
Tool toolHiLiteSize x + (2 * Tool toolIconSpacing) x @ toolPaletteSize).
        nodo add: (LookPreferences edgeDecorator on: aDrawingView)
                in: ((LayoutFrame new) leftOffset: 4 + Tool toolImageSize x + Tool
toolHiLiteSize x + (2 * Tool toolIconSpacing) x; topOffset: 0; rightFraction: 1; bottomFraction: 1).
        container := DependentComposite new.
        container add: buttonsNavegar borderedIn: (0 @ 0 corner: 0.25 @ 0.1); add: nodo in: (0
@ 0.1 corner: 1 @ 1).
        container add: buttonsHyper borderedIn: (0.25 @ 0 corner: 0.75 @ 0.1); add: nodo in:
(0 @ 0.1 corner: 1 @ 1).
        aWindow := ScheduledWindow new.
        aLabel := self armarLabel: primerNodo .
        aWindow label: aLabel.
        aWindow minimumSize: 500 @ (toolPaletteSize max: 280).
        aWindow component: container.
        aWindow open.
        HBrowser := aWindow.
        ^aDrawingEditor

    adelante
        "El usuario rquiere navegar hacia el nodo de adelante"

        (self navegador isTop or: self navegador isClear)
            ifFalse:
                [self cambiarNodo: self navegador sacaNodo.
                self refrescar]

    atras
        "El usuario rquiere navegar hacia el anterior nodo"

        (self navegador isBottom or: self navegador isClear)
            ifFalse:
```

```
                    [self cambiarNodo: self navegador popNodo.
                    self refrescar]

navegacion
        "El usuario rquiere ver el manager de navegación"

        HNavigationManagerBrowser openHCEditor: self

navegarA: unNodo
        "Navega a unNodo actualizandose. Todo nodo es presentado en una
        HPresentacionCompuesta"

        unNodo isNil
                ifFalse:
                        [self cambiarNodo: unNodo.
                        self navegador pushNodo: unNodo.
                        self refrescar]

navegarAObjeto: unObjeto
        "Navega al objeto"

        | unNodo |
        unNodo := unObjeto vistas detect: [:each | each isKindOf: HPresentacionCompuesta]
                                ifNone: [^nil].
        self navegarA: unNodo

navegarAsociacion
        "Abre un editor para la asociación seleccionada"

        | mc diccio unObjeto |
        mc := MenuBuilder new.
        diccio := self drawing propietario diccionarioDeAsociados.
        diccio keysAndValuesDo: [:key :value | mc addLabel: key asString value: value].
"mc addLabels: diccio keys andValues: diccio values."
        mc addLabel: 'Cancel' value: nil.
        unObjeto := mc startUp.
        (unObjeto isNil or: unObjeto = 0)
                ifFalse: [self navegarAObjeto: unObjeto]

navegarHFigure: aHFigure
        "Navega al nodo al cual la figura es la representación (area sensible)"

        | nodo |

        nodo := aHFigure siguienteNodo.
        self navegarA: nodo

propiedades
        "Abre un dialogo para editar las propiedes de la figura"

        self drawing propietario draw

reglas
        "El usuario rquiere ver las reglas de la metodología"

        HMetodologiaTool open

repositorio
        "Retorna la instancia gloaba del repositorio HCase. El propietario del nodo raiz es el
        repositorio"
```

```
        ^HCase
```

initializing

```
initialize
        "Inicializa el receptor"

        navegador := HNavigationManager new
```

MetaClass: HEditor class

defaults

```
drawingClass
        "Por defecto se abre un Drawing para crear proyectos"

        ^HProyectoManejadorDrawing
```

instance creation

```
abrir
        "Abre el resptor a partir del nodo raiz. "

        HBrowserSystem abrir

crear
        "Inicializa el receptor asociandole un HRepositorio para mantener los
        proyectos.
        HCEditor crear"

        HCase := RepositorioDeProyectos new initializeNombre:'Repositorio'.
        HBrowserSystem := self new. " initialize."
        HBrowserSystem navegador pushNodo:(HCaseManejadorDrawing newConPropietario:
HCase).
        HBrowserSystem abrir

new
        "Crea una nueva instancia del receptor"

        ^super new initialize

openOn: unDrawing
        ^self openOn: unDrawing withLabel: (unDrawing propietario nombre asString),' -
HProyecto '.

openOn: aDrawing withLabel: aLabel
        "Crea una nueva instacnia del receptor donde el primer nodo documento es aDrawing y
        su titulo es aLabel"

        | aDrawingView aToolPaletteView aWindow container toolPaletteSize aDrawingEditor
adelante atras editor buttonsNavegar buttonsHyper navegacion metodologia |
        aDrawingEditor := self new setDrawing: aDrawing.
        aDrawingView := DrawingView on: aDrawingEditor.
        aDrawingView setDrawing: aDrawing.
        aToolPaletteView := ToolPaletteView on: aDrawingEditor.
```

```
buttonsNavegar := CompositePart new.
adelante := (Button trigger) model: ((PluggableAdaptor on: aDrawingEditor)
                                        performAction: #adelante); label: '>>'.
atras := (Button trigger) model: ((PluggableAdaptor on: aDrawingEditor)
                                        performAction: #atras); label: '<<'.
buttonsNavegar add: atras in: (0 @ 0 corner: 0.5 @ 1).
buttonsNavegar add: adelante in: (0.5 @ 0 corner: 1 @ 1).
buttonsHyper := CompositePart new.
metodologia := (Button trigger) model: ((PluggableAdaptor on: aDrawingEditor)
                                        performAction: #metodologiaTool); label:
'Metodologia'.
navegacion := (Button trigger) model: ((PluggableAdaptor on: aDrawingEditor)
                                        performAction: #navegacion); label: 'Navegación'.
buttonsHyper add: navegacion in: (0 @ 0 corner: 0.5 @ 1).
buttonsHyper add: metodologia in: (0.5 @ 0 corner: 1 @ 1).
editor := CompositePart new.
toolPaletteSize := 280.
editor add: aToolPaletteView borderedIn: (0 @ 0 extent: 4 + Tool toolImageSize x +
Tool toolHiLiteSize x + (2 * Tool toolIconSpacing) x @ toolPaletteSize).
editor add: (LookPreferences edgeDecorator on: aDrawingView)
        in: ((LayoutFrame new) leftOffset: 4 + Tool toolImageSize x + Tool
toolHiLiteSize x + (2 * Tool toolIconSpacing) x; topOffset: 0; rightFraction: 1; bottomFraction: 1).
container := DependentComposite new.
container add: buttonsNavegar borderedIn: (0 @ 0 corner: 0.25 @ 0.1); add: editor in:
(0 @ 0.1 corner: 1 @ 1).
container add: buttonsHyper borderedIn: (0.25 @ 0 corner: 0.75 @ 0.1); add: editor in:
(0 @ 0.1 corner: 1 @ 1).
aWindow := ScheduledWindow new.
aWindow label: aLabel.
aWindow minimumSize: 500 @ (toolPaletteSize max: 280).
aWindow component: container.
aWindow open.
^aDrawingEditor
```

HFigure

```
Object ()
        VisualComponent ()
                VisualPart ('container')
                        Figure ('dependents' 'metaFigure')
                                CachedFigure ('cache' 'origin')

                                        HFigure ('propietario')
```

Class: HFigure

Superclass: CachedFigure
Category: HCase-Framework
Instance variables: propietario

Una HFigure representa un vista particular de un objeto, el tiene un puntero al ancor que representa, este tipo de figuras al ser cliqueadas con la herramienta de navegación, navegan hacia el destino marcado por el ancor.

Variables de instancia:
propietario -Un puntero a un concepto de la metodología, el cual debe ser una subclase de ConceptoDeJacobson en este prototipo.

accessing

```
drawing
        "Retorna el drawing que contiene la figura"

        ^container drawing

image: unaFigura
        "Asigna la figura del reseptor"

        cache := unaFigura asImage

origin: aPoint image: anImage
        "Asigna el origen de la imagen y la imagen de la figura con anImage"

        self origin: aPoint.
        self image: anImage.
        ^self

propietario
        "Retorna el propietario del receptor"

        ^propietario

propietario: unConcepto
        "Retorna el reseptor. Asigna el objeto de la metodología al reseptor quedando
        este como dependiente"

        propietario isNil ifFalse: [propietario eliminarVista: self].
        propietario := unConcepto.          "propietario agregarVista: self"
        propietario addDependent: self
```

```
propietarioDirecto
        "Se define como propietario directo al propietario"

        ^propietario

siguienteNodo
        "Retorna el siguiente nodo del grafo estático"

        ^self propietario vistas detect: [:each | each isKindOf: HPresentacionCompuesta]
                ifNone: [^nil]
```

testing

```
isHCompositeFigure
        "Retorna false pues no pertenece a las HCompositeFigure"

        ^false

isHFigure
        "Retorna true"

        ^true
```

navigating

```
pressNavigeitorTool
        "Envia al Manejador de Hypermedia una HFigure (area sensible) para ser navegado en
el
        contexto presente."

        HBrowserSystem navegarHFigure: self
```

transforming

```
growBy: aPoint
        "HFigure can't grow"

scaleBy: aPoint
        "HFigure cant be scaled."

transformBy: aTransformation
        "No transformation permitted on HFigures"
```

alining

```
alinearVerticalA: unSmallInteger
        "Alinea verticalmen a unSmallInteger por el centro
        del reseptor"

        | diferencia |
        diferencia := unSmallInteger - self center x.
        self translateBy: diferencia @ 0
```

updating

update: anAspectSymbol
"Actualiza el receptor lugo de que un objeto dependiente se modifica"

^self perform: anAspectSymbol

removing

hardRemove
"El contenedor implementa el método para remover definitivamente el receptor"

self propietario remove

propietario changing

changedDescriptor
"El receptor no puede cambiar"

MetaClass: HFigure class

instance creation

origin: aPoint

^self new origin: aPoint image: self figura

origin: aPoint figure: aImage

^self new origin: aPoint image: aImage

resources

figura

"Figura que representa el propietario"

HPresentacionSimple

Object ()
VisualComponent ()
VisualPart ('container')
Figure ('dependents' 'metaFigure')
CompositeFigure ('figures' 'visibleArea' 'visibleFigures' 'showVisibleArea')

HPresentacionSimple ()
ActorFigure ()
AtributoFigure ()
CasoDeUsoFigure ()
ControlFigure ()
EntidadFigure ()
EspecificacionFigure ()
InterfazFigure ()
ModeloDeAnalisisFigure ()
ModeloDeCasosDeUsoFigure ()
ModeloDeDiseñoFigure ()
ObjetoDelRepositorioFigure ()
ProyectoFigure ()
RepositorioDeProyectosFigure ()
SubsistemaFigure ()
TarjetaDeClaseFigure ()

Class: **HPresentacionSimple**

Superclass: CompositeFigure
Category: HCase-Framework

Modela el comportamiento de las representaciones gráficas de los conceptos de la metodología. Instancias de esta clase son figuras compuesta (Figure). Internamente manejan un conjunto de Handle para conectarse con demás figuras y un menú para modificar sus propiedades.

La componente principal es una colección de figuras simples (instancia de HFigure y HTextLabrlFigure) la cual es heredada de CompositeFigure (detalles de ésta clase son especificados en HotDraw).

Subclases de esta clase deben ser agregadas para implementar conceptos de la metodología que tengan una presentación gráfica en algún diagrama.

HPresentaciónSimple agregas caracteristicas hypermediales a las componentes grágicas. Contiene subfiguras que son areas sensibles anclada a diferentes nodos.

Subclases debem implementar los siguientes métodos de clase:

- principalFigure
- icanTool

Subclases deben implementar los siguientes metodos de instancias:

- classPropitario
- classPrincipalFigure

displaying

```
displayOn: GC
        "Redibuja las subfiguras que contiene sobre un GC (GraphicContext)"

        visibleFigures reverseDo: [:each | each displayOn: GC].
        ^self
```

propiedades
"Abre un dialogo para editar las propiedes de la figura"

self propietarioDirecto draw

accessing

cambiarLabel
"Permite al usuario que modifique el label. Cambiar el nombre del propietario"

self propietarioDirecto nombre: (Dialog request: 'Nombre:' initialAnswer: self label)

drawing
"Retorna el drawing que contiene a la figura"

^container drawing

editarPrincipalFigure
"Edita la figura para modificar el aspecto visual de la misma.

UIMaskEditor new openOnClass: self principalFigura class andSelector: #figura"

HPresentacionTool openRepresentacion: self propietarioDirecto class name

estableserRelacion: unSymbolRelacion entreOrigen: unOJOrigen yDestino: unOJDestino
"Retorna el reseptor. Actualiza el repositorio para que contenga la relación entre los argumentos (si es que no existe)"
"RepositorioDeProyecto verificarRelacion: unaRelacion entreOrigen: o yDestino: d"

```
ArcoConcreto
        destino: unOJDestino
        origen: unOJOrigen
        tipo: unSymbolRelacion
        label: unSymbolRelacion asString
```

handles
"Retorna los handles y los adiciona a la lista del receptor. Por defecto toda las figuras del HCase tienen un ConectionHandle"

```
^(super handles) add: ((HConnectionHandle
                on: self
                at: #centerFigurePrincipal
                class: HArrowFigure)
                action: [:cu :figure | cu asociar: figure]); yourself
```

label
"El label de la figura es una subfigura de la cual se retorna el string que la conforma. Por defecto el label es la segunda subfigura de la colección"

^(self labelFigure) string

label: aString
"Asigna el label al receptor, la cual es la segunda subfigura de la colección"

```
| labelFigure |
labelFigure := self labelFigure.
labelFigure string: aString.
self changed
```

```
labelFigure
        "Por defecto el label es la segunda figura de la colección"

        ^(self figures at: 2)

locator
        "Retorna la ubicación por defecto donde se consentran los destinos de las conecciones
        (flechas)"

        ^Locator on: self at: #center

menu
        "Retorna el menú que se muestra cuando se presiona el boton derecho sobre el
        receptor"

        ^PopUpMenu
                labels: 'propiedades\editar figura\cambiar label' withCRs
                lines: #(1 )
                values: #(#propiedades #editarPrincipalFigure #cambiarLabel )

principalFigura
        "Retorna la subfigura princial de las componentes del receptor. Si no está asignada la
        crea y la añade como la primera de la colección"

        figures isNil ifTrue: [^HFigure origin: 0 @ 0 figure: self class principalFigure].
        ^figures at: 1

propietarioDirecto
        "Retorna el propietario de la subfigura principal, el cual se define como propietario
        directo"

        ^self principalFigura propietarioDirecto

representaciones
        "Retorna la lista de HFigures que componen el receptor"

        ^figures

userOJ
        "Permite que el espectador ingrese el identificador del Objeto
        de la metodologia"

        | nombre |
        nombre := Dialog request: 'Nombre del ' , self classPropietario name.
        nombre = '' ifFalse: [^self classPropietario new nombre: nombre].
        ^nil
```

navigating

```
pressNavigeitorTool
        "Las figuras compuestas no se navegan. Se navegan las componetes"

pressNavigeitorToolAtPoint: aPoint
        ""

        | aFigure |
        visibleFigures
                do:
                        [:each |
                        aFigure := each figureAt: aPoint.
```

```
                    aFigure notNil ifTrue: [aFigure pressNavigeitorTool]].
        ^self
```

testing

```
isHCompositeFigure
        ^true

isHFigure
        ^true
```

initializing

```
classDrawing
        "Retorna la clase del nodo por defecto"

        ^HObjetoDelRepositorioDrawing

classPrincipalFigure
        "Retorna la clase (subclase de HFigure) de la figura principal de la representación"

        self subclassResponsibility

classPropietario
        "Retorna la clase del propietario de la figura. Cada subclase debe implementar este
        mensaje. Esta debe ser una clase del nivel de Objetos en el repositorio"

        self subclassResponsibility

initializeCon: unOJ at: aPoint
        "Arma una figura que reprecenta al reseptor"

        | textLabel principalHFigure figuras  classNodo |
        principalHFigure :=HFigure origin: aPoint figure: self class figuraPrincipal.
        textLabel := HTextLabelFigure string: unOJ nombre asString at: principalHFigure
bottomLeft.
        principalHFigure propietario: unOJ.
        textLabel propietario: unOJ.
        figuras := OrderedCollection new.
        figuras add: principalHFigure; add: textLabel.
        self setFigures: figuras.
        self alinearVertical.
        "Crea el nodo del repositorio vacio - un Drawing"
        (classNodo := self classDrawing) isNil ifFalse: [classNodo newConPropietario: unOJ].
        ^self

initializeInPoint: aPoint inDrawing: aDrawing
        "Dado aPoint y aDrawing agrega una figura en tal posición cuyo label es dado por el
        usuario"

        | unOJ representacion |
        unOJ := self userOJ.
        unOJ isNil
                ifFalse:
                        [representacion := self initializeCon: unOJ at: aPoint.
                        representacion
                                estableserRelacion: #ParteDe
                                entreOrigen: unOJ
                                yDestino: aDrawing propietario.
```

```
                    ^representacion].
        ^nil
```

transformation

```
growBy: aRectangle
        "CompositeHFigures no pueden ser expandidas"

aligning

alinearVertical
        "Alinea verticalmente las figuras visibles del reseptor"

        | xAlineacion |
        xAlineacion := self center x.
        visibleFigures do:[:each | each alinearVerticalA: xAlineacion.].
        self resetVisibleArea.
```

bounding box accessing

```
centerFigurePrincipal
        "El centro de la figura se indica por el centro de la figura principal que es la primera de
la
        colección. Si no tiene es el centro del grupo de figuras"

        | figure |
        figure := self principalFigura center.
        figure isNil
                ifTrue: [^figure center]
                ifFalse: [^self center]

locatorCenter
        "El centro de la figura se indica por el centro
        de la figura principal que es la primera de la colección"

        ^ (visibleFigures at: 1) center
```

case-propertie

```
proyecto
        "Retorna el actual proyecto que lo contiene"

        ^self container proyecto
```

private

```
setFigures: aCollection
        "Asigna aCollection a la variable de instancia figures del receptor"

         self setFigures: aCollection visibleArea: (aCollection inject: aCollection first
displayBox into: [:rect :each | rect merge: each displayBox]).
        self showVisibleAreaIndicator
```

asociating

```
asociar: unaCompositeHFigure con: aSymbolAsociacion
        "Retorna una ArcoConcreto que rpresenta una asociación entre el representate del
receptor y el de unaCompositeHFigure"

        ^(ArcoConcreto
                destino: unaCompositeHFigure propietarioDirecto
                origen: self propietarioDirecto
                tipo: aSymbolAsociacion)

nombreDeAsociacionesCon: figureTarget
        "Retorna una colección con los nombre de las asociaciones entre los propietarios del
        reseptor y figureTarget"

        ^MetodologiaOOSE relacionesEntre: self propietarioDirecto class y: figureTarget
propietarioDirecto class

userAsociarA: figureTarget
        "Retorna un ArcoConcreto desde la asociación selecionada por el usuario. Muestra un
        menú con los tipos de asociaciones entre los propietario del reseptor y figureTarget. Si
no
        existen asociaciones entre los objetos retorna nil"

        | nombreDeAsociaciones asociacionName labels |

        nombreDeAsociaciones := self nombreDeAsociacionesCon: figureTarget.
        nombreDeAsociaciones isEmpty
                ifFalse:
                        [labels := Array new: 2.
                        labels at: 1 put: nombreDeAsociaciones asArray.
                        labels at: 2 put: #('cancelar' ).
                        nombreDeAsociaciones addLast: 'cancelar'.
                        asociacionName := (PopUpMenu labelList: labels values:
nombreDeAsociaciones asArray) startUp.
                        (asociacionName isNil or: [asociacionName = 'cancelar'])
                                ifFalse: [^self asociar: figureTarget con: asociacionName
asSymbol]].
        ^nil
```

checking

```
checkAsociar: figureTarget con: asociacion
        "Cheque si es correcta la asociacion del reseptor con la figureTarget"
        " Regla de chequeo"
        ^true
```

removing

```
delete
        "Elimina el receptor del repositorio"

        self propietarioDirecto remove.
        super delete

hardRemove
        "Indica al contenedor que el receptor debe removerse"

        self propietarioDirecto remove
```

metodologia

```
metodologia
	"Retorna la metodologia corriente"

	^MetodologiaOOSE
```

MetaClass: *HPresentacionSimple class*

default

```
creationTool

	^HCreationTool
		icon: self iconTool
		cursor: self cursorTool
		class: self

cursorTool
 ^Cursor origin

toolPalette
	^MappedPalette with: ColorValue white with: ColorValue red
```

instance creation

```
createNotifying: aView

	aView sensor waitButton.
	^self new initializeInPoint: (aView sensor cursorPoint)  inDrawing: aView drawing
```

resources

```
figuraPrincipal
	"UIMaskEditor new openOnClass: self andSelector: #figuraPrincipal"

	^Image extent: 40@40 depth: 1 palette: CoveragePalette monoMaskPalette bits:
(ByteArray fromPackedString:
'@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@b') pad: 32

iconTool
	"UIMaskEditor new openOnClass: self andSelector: #iconTool"
```

^Image extent: 16@16 depth: 1 palette: CoveragePalette monoMaskPalette bits: #[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0] pad: 32

HPresentacionCompuesta

```
Object ()
        Model ('dependents')
                Drawing ('figures' 'damagedRegion' 'pageResolution')

                        HPresentacionCompuesta ('propietario')
                                HCaseManejadorDrawing ()
                                HObjetoDelRepositorioDrawing ()
                                HProyectoManejadorDrawing ()
                                MADrawing ()
                                MCUDrawing ()
                                MDDrawing ()
                                SistemaDrawing ()
```

Class: **HPresentacionCompuesta**

Superclass: Drawing
Category: HCase-Framework
Instance variables: propietario

HPresentacionCompuesta representa contenedores de figuras que expresan un diagrama o un documento gráfico de la metodología.

La clase agrega al comportamientop de Drawing las responsabilidades de comunicación y conexión con el propietario (concepto de la metodología) que reside en el repositorio.

Las variables de instancia son:

propietario - NodoDelRepositorio. Cada instancia mantiene un puntero al nodo del repositorio que representa.

update

```
actualizar: unSymbol con: unParametro y: otroParametro
        "self subclassResponsibility"
```

accessing

```
drawing

        ^self

propiedades
        "Abre un dialogo para editar las propiedes de la figura"

        HM
                abrir: (ArcoVirtual
                                destino: self propietario
                                origen: self propietario
                                tipo: #self)
                presentacion: #Tarjeta
                drawing: self
                claseDeEditor: HCaseManejador.

propietario
```

```
        "Retorna el propietario del reseptor "

        ^propietario

    propietario: unNodoDeRepositorio
        "Asigna el propietario del receptor el cual normalmente debería ser un nodo del
        repositorio."

        propietario isNil ifFalse: [propietario eliminarVista: self].
        propietario := unNodoDeRepositorio.
        unNodoDeRepositorio addDependent: self.
        unNodoDeRepositorio agregarVista: self
```

initializing

```
    initializeCon: unOJ
        "Inicializa el reseptor con propietario unOJ"

        self initialize.
        propietario := unOJ.
        unOJ addDependent: self.
        unOJ agregarVista: self.
        ^self

    installInEditor: anEditor
        "Instala en anEditor las herramientas correspondientes a la clase del receptor."

        ^anEditor tools: self class tools



    hardRemove: aFigure
        "Remueve del propietario el objeto propietario del argumento. "
        "self remuve: aFigure."

        self propietario remove: aFigure propietarioDirecto.
        ^aFigure
```

propietario changing

```
    changedDescriptor
        "El receptor no puede cambiar"
```

damaging

```
    update: aFigureOAspect
        "Actualiza el receptor luego de que algún cambio ocurrio en alguno de sus
dependientes"

        (aFigureOAspect isKindOf: Figure)
            ifTrue: [self damageRegion: aFigureOAspect displayBox]
            ifFalse: [self perform: aFigureOAspect]
```

MetaClass: HPresentacionCompuesta class

intance creation

```
newConPropietario: unPropietario
        " Crea una instancia del reseptor y la inicializa con un propietario"

        ^self new initializeCon: unPropietario.
```

default

```
classPropietario
        "Retorna la clase del objeto de la metodología donde el reseptor es el propietario"

        self subclassResponsibility

emptyTools
        "Retorna las tools en blanco"

        ^(OrderedCollection new)

tools
        "Retorna las tools comunes a los documentos de HCase"

        ^(self emptyTools) add: NavigeitorTool new;
                           add: SelectionTool new;
                           add: ScrollingTool new;
                           add: FigureActionTool delete;
                           add: FigureActionTool hardRemove;
                           add: TextFigure creationTool; yourself
```

Tool

```
Object ()
        VisualComponent ()
                Tool ('icon' 'cursor' 'controller')

                        HCreationTool ('className')
                        HNavigeitorTool ()
```

Class: HCreationTool

Superclass: Tool
Category: HCase-Draw
Instance variables: className

HCreationTool especializa Tool para proporcionar caracteristicas especiales para HCase. Esta es una herramienta para creación de subclases de HFigures. Cada invocación de una instancia de esta clase creará una figura dentro del correspondiente Drawing. La HFigure a crear está dada por la variable de instancia className.

Variables de instamcias:

className : Symbol. Es el nombre de la clase de la HFigura que se creará por medio de su invocación.

invoking

```
pressBackground
        "Crea la figura correspondiente dentor del drawing en la cual fue precionada"

        self createFigure

pressFigure: aFigure
        "Crea una HFigure primera en la lista de figuras"

        self createFigure
```

private

```
createFigure
        "Crea una subclase de HFigura y la adiciona al drawing "

        | aFigure |
        aFigure :=self creationClass createNotifying: self view.
    aFigure notNil ifTrue: [self view addFigure: aFigure].
        ^aFigure

creationClass
        "Retorna la clase dada por className"
    ^Smalltalk at: className

setClassName: aSymbol
        "Asigna la variable de instancia className con aSymbol"

        className := aSymbol
```

MetaClass: ***HCreationTool class***

instance creation

```
cursor: aCursor class: aClass
        "Crea una instancia del receptor cuyo cursor es aCursor y cada invocación de la
        herramienta creará una HFigure de clase aClass."

        ^(self cursor: aCursor)
                setClassName: aClass name

icon: anImage cursor: aCursor class: aClass
        "Crea una instancia del receptor cuyo cursor es aCursor y cada invocación de la
        herramienta creará una subclase de HFigure de clase aClass, y su imagen en la paleta de
        herramientas está dada por anImage"

        ^(self icon: anImage cursor: aCursor)
                setClassName: aClass name
```

Class: HNavigatorTool

Superclass: Tool
Category: HCase-Draw

HNavigatorTool es un herramienta especial para navegar al propietario de la presentación, compueta o simple.

invoking

```
press
        | figure point |
        point := self sensor cursorPoint.
        ((figure := self figureAtPoint: point ) isNil or: [figure isHCompositeFigure not])
                ifFalse: [(figure pressNavigeitorToolAtPoint: point) ].
        ^self
```

MetaClass: ***HNavigatorTool class***

instance creation

```
new
        "Crea una nueva instancia del receptor"

        ^self icon: (Image
                        extent: 16 @ 16
                        depth: 1
                        palette: (MappedPalette with: ColorValue red with: ColorValue white)
                        bits: #[255 255 255 255 135 225 231 231 243 207 249 159 252 63
252 63 252 63 252 63 249 159 243 207 231 231 135 225 255 255 255 255 ]
                        pad: 8)
```

cursor: Cursor thumbsUp

HMetodologia

Object ()

HMetodologia ('nombreDeLaMetodologia' 'reglas' 'relaciones' 'presentaciones')
MetodologiaDeJacobson ()

Class: HMetodologia

Superclass: Object
Category: HCase-Framework
Instance variables: nombreDeLaMetodologia reglas relaciones
presentaciones

HMetodologia es una clase abstracta que mantiene caracteristicas genericas de las metodologías. Dado que HCase permite ser instanciado para diferentes metodologías esta clase modela las principales caracteristicas de las metodologías. Subclase deben modelar metodologías concretas (Ej MetodologiaDeJacobson).

Varialbles de instancias:
nombreDeLaMetodologia :String. El nombre de la metodología instanciada.
reglas :Set. Se tiene ternas (conceptoOrigen, conceptoDestino, relacion) que determinan las relaciones válidas entre los diferentes conceptos de la metodología.
relaciones :Dictionary. asociacion: (key: symbolRelacion, value: symbolInversa). Lista de relaciones entre conceptos soportada por la metodología, con su correspondiente inversa.
presentaciones :Dictionary. Asociacion: (key: ConceptoClass, value: HPresentacionTemplate) cada concepto de la metodología se le asocia un template que indica a HCase la forma que este es presentado al espectador.

private

```
presentaciones: unDicDePresentaciones
        "Private - Asigna el diccionari de presentaciones"

        presentaciones := unDicDePresentaciones

reglas: unSetDeReglas
        "Private - Asigna el Set de reglas de la metodología"

        reglas := unSetDeReglas

relaciones: unDicDeRelaciones
        "Private - Asigna el diccionario de las relaciones soportadas por la metodología"

        relaciones := unDicDeRelaciones

superClassDeConceptos
        self subclassResponsibility
```

initialize

```
initialize
        "Inicializa el receptor con toda sus variables de instancias vacias"
```

```
        reglas := Set new.          "se tiene ternas de (conceptoOrigen, conceptoDestino,
relacion)"
        relaciones := Dictionary new.       "asociacion: (key: ConceptoClass, value:
HPresentacionTemplate)"
        presentaciones := Dictionary new
```

accessing

```
addPresentacion: unConcepto template: unaPresentacionTemplate
        "Asocia un template a un concepto"

        presentaciones at: unConcepto put: unaPresentacionTemplate

addRegla: unaRegla
        "Adiciona una regla a la metodología"

        reglas add: unaRegla

addRelación: unSymbol inversa: otroSymbol
        "Adiciona una relación con su inversa al receptor"

        relaciones at: unSymbol put: otroSymbol

conceptoAñadir: aClassName
        "Adicona un concepto a la metodología. Agrega una nueva clase como subclase del
        concepto determinado por la metodología"

        | conceptoClass |
        conceptoClass := self classConcepto "(Ej ConceptoDeJacobson)"
                                subclass: aClassName asSymbol
                                instanceVariableNames: "
                                classVariableNames: "
                                poolDictionaries: "
                                category: 'HCase-Rep-Support'.
        ^conceptoClass

conceptoClass: aClassName
        "Retorna la superclase de los conceptos"

        ^Smalltalk at: aClassName

conceptoEliminar: aClassName
        "Elimina un concepto del repositorio siempre que no tenga subclases"

        | class |
        class := Smalltalk at: aClassName.
        class subclasses size > 0 ifTrue: [self notify: aClassName , ' tiene subclases'].
        class removeFromSystem

conceptos
        "Retorna una colección con los conceptos retornados por el receptor"

        ^self superClassDeConceptos allSubclasses

conceptosNombres
        "Retorna una colección con los nombre de los conceptos del receptor"

        ^self conceptos collect: [:class | class name]
```

```
        inversa: unSymbol
                "Dada una relación soportada por el receptor, retorna la relación inversa"

                ^relaciones at: unSymbol
                        ifAbsent:
                                [Dialog warn: 'La relación no es soportada por la metodología.'.
                                nil]

        nombreDeLaMetodologia
                "Retorna el nombre de la metodología instanciada por el receptor"

                ^nombreDeLaMetodologia

        nombreDeLaMetodologia: aString
                "Asigna el nombre de la metodología"

                nombreDeLaMetodologia := aString

        reglas
                "Private - Retorna el conjunto de reglas soportados por el receptor"

                ^reglas

        reglasConRelacion: unSymbol
                "Retorna una colección de reglas cuyo nombre de la relación corresponde con
                unSymbol"

                | reglasEncontradas |
                reglasEncontradas := reglas select: [:regla | regla relacion = unSymbol].
                ^reglasEncontradas

        reglasEntre: classConceptoOrigen y: classConceptoDestino
                "Retoran una colección de reglas en donde el concepto origen y destino coinciden con
                los parametros"

                | reglasEncontradas |
                reglasEncontradas := reglas select: [:regla | regla origen = classConceptoOrigen and:
[regla destino = classConceptoDestino]].
                ^reglasEncontradas

        reglasParaDestino: classConcepto
                "Retorna una colección con las reglas en donde el concepto destino coincide con
                classConcepto"

                | reglasEncontradas |
                reglasEncontradas := reglas select: [:regla | regla destino = classConcepto].
                ^reglasEncontradas

        reglasParaOrigen: classConcepto
                "Retorna una colección con las reglas en donde el concepto origen coincide con
                classConcepto"
                "Ej MetodologiaOOSE reglasParaOrigen: Actor"

                | reglasEncontradas |
                reglasEncontradas := reglas select: [:regla | regla origen = classConcepto].
                ^reglasEncontradas

        reglasParaOrigen: classConcepto destino: otroClassConcepto relacion: aSymbol
                "Retorna una colección con las reglas que cumplen con cocepto origen, destino y
                nombre de relación indicado por los parámetros"
```

```
                reglas select: [:regla | (regla origen = classConcepto and: [regla destino =
otroClassConcepto and: [regla relacion = aSymbol]])
                                        ifTrue: [^regla]]

        relaciones
                "Private - Retorna el diccionario de relaciones"

                ^relaciones

        relacionesEntre: classConceptoOrigen y: classConceptoDestino
                "Retoran una colección de relaciones en donde la clase del concepto origen y detino
                se corresponde con los parámetros"
                "Ej MetodologiaOOSE relacionesEntre: Actor y: CasoDeUso"

                | relacionesEncontradas |
                relacionesEncontradas := (self reglasEntre: classConceptoOrigen y:
classConceptoDestino)
                                                collect: [:regla | regla relacion].
                ^relacionesEncontradas asOrderedCollection

        relacionesNombres
                "Retorna una colección con el nombre de las relaciones soportadas por el receptor"

                ^relaciones keys
```

MetaClass: ***HMetodologia class***

HNavigationManager

Object ()
Model ('dependents')

HNavigationManager ('pilaDeNodos' 'indice' 'top' 'bottom' 'usuario')

Class: **HNavigationManager**

Superclass: Model
Category: HCase-Framework
Instance variables: pilaDeNodos indice top bottom usuario

HNavigationManager modela el conjunto de responsabilidades elementales para el manejo de un entorno hipermidial.

Varables de instancias:

pilaDeNodos - Estructura para manejar la pila de nodos recoridos por el espectador
indice - Nodo actual del hiperespacion.
top - Indica si el nodo actual está en el tope de la pila
bottom - Indica si el nodo actual es el último de la pila
usuario - Espectador actual del hiperespacio.

apertura

```
abrir
        "Abre una instancia del receptor"
```

accessing

```
usuario
        " Retorna el usuario actual "

        ^usuario

usuario: unUsuario
        " Setea unUsuario como el actual usuario del reseptor. Todo las
        trabsacciones realizadas decaeran en su responsabilidad"

        usuario := unUsuario
```

initialize-release

```
initialize
        "Inicializa el receptor limpiando la pila de nodos recorridos"

        self clearPila.
        usuario := Usuario new
```

pila

```
clearPila
        "Vacia la pila de nodos recoridos del receptor"
```

```
        pilaDeNodos := OrderedCollection new.
        indice := 0.
        top := true.
        bottom := false

isBottom
        "Retorna true si se encuntra al fina de la pila"

        ^indice = 1

isClear
        "Retorna true si la pila del receptor está vacía"

        ^self pilaSize = 0

isTop
        "Retorna true si el nodo actual esta al tope de la pila"

        ^indice = self pilaSize

nodoAt: unIndice
        "Retoran el nodo indicado por unIndice"

        self isClear ifTrue: [^nil].
        (self pilaSize >= unIndice and: [unIndice > 0])
                ifTrue: [^pilaDeNodos at: unIndice]
                ifFalse: [^nil]

pilaDeNodos
        "Retorna la pila de nodos del receptor"

        ^pilaDeNodos

pilaSize
        "Retoran la cantidad de nosdos recorridos"

        ^pilaDeNodos size

popNodo
        "Saca el nodo del tope de la pila"

        self isClear ifTrue: [^nil].
        self isBottom ifFalse: [indice := indice - 1].
        ^pilaDeNodos at: indice

posicionarseEn: unIndice
        "Se posiciona en el nodo indicado por un índice y lo retorna"

        self isClear ifTrue: [^nil].
        (self pilaSize >= unIndice and: [unIndice > 0])
                ifTrue: [indice := unIndice]
                ifFalse: [^nil].
        ^pilaDeNodos at: indice

pushNodo: unNodo
        "Agrega un nodo al tope de la pila"

        pilaDeNodos addLast: unNodo.
        indice := self pilaSize
```

```
sacaNodo
        "Retora el ultimo nodo y mueve el puntero al anterior. Inversa del metodo popNodo:"

        self isClear ifTrue: [^nil].
        self isTop ifFalse: [indice := indice + 1].
        ^pilaDeNodos at: indice
```

MetaClass: HNavigationManager class

instance creation

```
new
        "Crea una instancia del receptor"

        ^super new initialize
```