Mecanos Reutilizables Estáticos

Versión 0.9. Marzo de 1998

TR-GIRO-01-98V0.9

Francisco José García Peñalvo [1]
e-mail: fgarcia@ubu.es

José Manuel Marqués Corral [2]
e-mail: jmmc@infor.uva.es

Miguel Ángel Laguna [2]
e-mail: mlaguna@infor.uva.es

Jesús Manuel Maudes Raedo [1]
e-mail: jmaudes@ubu.es

Resumen

La reutilización no supone un nuevo concepto en el desarrollo del software. Se ha estado reutilizando bibliotecas de software durante mucho tiempo, pero viendo al código fuente como el único objeto de la reutilización y generalmente empleando técnicas intuitivas. Sin embargo, actualmente, el concepto de reutilización ha evolucionado hacia la idea de que todo el conocimiento y los productos derivados de la producción de software son susceptibles de ser reutilizados en la construcción de nuevos sistemas, surgiendo de esta forma el concepto de asset o de componente software reutilizable.

No obstante, aunque el ámbito de la reutilización se ha extendido hacia las fases iniciales del ciclo vital de los desarrollos software, las diferentes clasificaciones de assets siguen sin contemplar una reutilización en diferentes niveles de abstracción de forma simultánea. Este tipo de reutilización multinivel favorecería la idea de reutilización sistemática desde la especificación de requisitos a la implementación.

El presente documento presenta una estructura de reutilización compleja que da soporte a un elemento software reutilizable definido en diferentes niveles de abstracción y que permite que tanto el desarrollador para reutilización como el desarrollador con reutilización tengan una visión más cercana a la reutilización de subsistemas a la hora de ponerla en práctica, cada uno desde la perspectiva propia del rol que desempeña.

Palabras Clave

Reutilización de software, estructuras complejas de reutilización, relaciones semánticas, assets.

[1] Departamento de Ingeniería Electromecánica y Civil. Área de Lenguajes y Sistemas Informáticos. Escuela Universitaria Politécnica, Universidad de Burgos. Av. General Vigón S/N, 09006 Burgos (España).

[2] Departamento de Informática. Edificio de Tecnologías de la Información y las Telecomunicaciones. Universidad de Valladolid. Paseo del Cementerio S/N, 47011 Valladolid (España).

Tabla de Contenidos

Tabla de Ilustraciones

1. Introducción

La reutilización sistemática del software es la alternativa de desarrollo de software que puede producir una mejora significativa en la productividad y en la calidad del software [Biggerstaff, 1992]. Pero la reutilización es más compleja de llevar a cabo que otras tecnologías de desarrollo de software por diversos motivos, entre los que cabría destacar los siguientes hechos:

- *La desconfianza en el trabajo realizado por otras personas* [McClure, 1997].
- *La falta de soporte que de la reutilización hacen las metodologías de desarrollo* [McClure, 1997].
- *La inversión inicial que requiere la reutilización del software es un coste extra en los proyectos* [Karlsson, 1995].
- *Para obtener un retorno más adecuado de la inversión realizada en la reutilización del software, ésta debe afectar a elementos de mayor nivel abstracción que el código fuente, con toda la problemática y complejidad que ello conlleva* [McClure, 1997].

Inicialmente, la reutilización se concibe sólo en el nivel de implementación, tradicionalmente mediante el uso de bibliotecas de funciones, caracterizándose por llevarse a cabo con una total carencia de tintes metodológicos.

Actualmente, el concepto de reutilización ha evolucionado hacia la idea de que todo el conocimiento y productos derivados de la producción de software son susceptibles de ser reutilizados en la construcción de nuevos sistemas software [Freeman, 1987a]. El aumento de la potencia de la reutilización pasa por el aumento del nivel de abstracción donde se aplica. La abstracción constituye un elemento fundamental en la reutilización de software, así David Parnas [Parnas et al., 1989] afirma que el desarrollo y clasificación de abstracciones incrementa las posibilidades de reutilización del software desarrollado, ya que los desarrollos de una abstracción pueden ser reutilizados para cualquier modelo válido de la abstracción. Así, la reutilización debería enfocarse hacia la reutilización de requisitos y diseños de alto nivel.

Surge así un nuevo concepto para reflejar el alcance actual de la reutilización, ***el asset***. Un asset se define como *cualquier producto del ciclo de vida del software que pueda ser potencialmente reutilizado. Esto incluye: modelo de dominio, arquitectura de dominio, requisitos, diseño, código, bases de datos, esquemas de bases de datos, documentación, manuales de usuario, casos de prueba...* [DOD, 1992], [NIST, 1994].

Los assets de mayor nivel de abstracción son los potencialmente más beneficiosos para la reutilización del software. Esto es debido a que las primeras fases de los desarrollos software constituyen el mayor esfuerzo de todo el desarrollo, por tanto, los assets generados en estas fases deben suponer un importante aumento de la productividad y un ahorro significativo. Además, los beneficios de la reutilización de estos assets se verían aumentados si se reutilizasen los assets subsiguientes derivados de estos assets de alto nivel de abstracción [Cybulski et al., 1997a]. Para el estudio de los assets propios de las primeras fases del ciclo de desarrollo se ha creado un grupo de trabajo en el workshop WISR8[1].

[1] Este grupo está liderado por ***Jacob L. Cybulski***, y trabaja en la identificación, clasificación y reutilización de los assets generados en las primeras etapas de los desarrollos software, esto es, concepción del software, análisis de requisitos, estudio de viabilidad, especificación, diseño arquitectónico y diseño detallado [Edwards and Weide, 1997].

El aumento del ámbito de la reutilización ha quedado plasmado en las diferentes clasificaciones de los elementos software reutilizables que se pueden encontrar en la literatura, entre las cuales se pueden citar [Jones, 1984], [Biggerstaff, 1992], [Krueger, 1992], [Mili et al., 1995], [Edwards et al., 1997].

No obstante, aunque el aumento del ámbito de la reutilización es positivo, la totalidad de las taxonomías anteriormente mencionadas presentan a los assets definidos en un solo nivel de abstracción o correspondiéndose a una sola fase del ciclo de vida de desarrollo del software. Como consecuencia de esto, ni los desarrolladores para reutilización, ni los desarrolladores con reutilización tienen concepción de la reutilización sistemática que puede afectar a varios niveles de abstracción al mismo tiempo.

Para extender el concepto de reutilización hacia una visión de alcance multinivel, se define una estructura compleja de reutilización que dé origen a un elemento software reutilizable definido en diferentes niveles de abstracción, y que va a recibir el nombre de ***MECANO***.

El inicio del trabajo de definición de esta estructura de reutilización compleja coincide con la creación del grupo GIRO (*Grupo de Investigación en Reutilización y Orientación al Objeto*) en noviembre de 1996.

En el presente documento se pretende recoger el trabajo realizado hasta el momento en aras de la definición de los mecanos desde un punto de vista informal y estático [García et al., 1997a], [García et al., 1998].

El resto del documento se organiza como sigue. En la sección 2 se plantean las nociones básicas sobre las que debe incidir una estructura compleja de reutilización, haciendo un pequeño estado del arte de cada una de ellas. En la tercera sección se presenta la definición del concepto de mecano reutilizable, así como la de sus tipos. En esta sección se presentará una lista con los requisitos que debe cumplir esta estructura. La sección 4 está dedicada por completo a la descripción de los mecanos estáticos, realizando un modelo que represente su estructura. Las conclusiones y el trabajo futuro cierran este documento en la sección 5.

2. Definiciones básicas

El objetivo del presente documento es abordar la definición de una estructura compleja de reutilización que abarque diferentes niveles de abstracción. Pero, previamente se va a proceder a enunciar de una forma intuitiva[2] los conceptos que servirán de base para dicha definición. Estas definiciones básicas vendrán acompañadas de un estado del arte, en el que se refleje las principales aportaciones en cada una de las parcelas estudiadas. Todas estas referencias ayudaran a enunciar la definición de lo que es un mecano, así como a determinar aquellos requisitos funcionales y de sistema que debe cumplir esta estructura de reutilización.

2.1 Desarrollo software

El objetivo de una estructura compleja de reutilización, al menos en un primer momento, se centra en soportar el esfuerzo invertido en un desarrollo software concreto.

[2] Las definiciones aquí presentadas intentan reflejar los conceptos que sustentan la base sobre la que se va a definir la estructura estática de reutilización. El enunciado de las mismas se ha hecho desde el punto de vista marcado por la naturaleza intrínseca del trabajo en curso, recibiendo, como es natural, influencias de la literatura, pero sin ajustarse de forma estricta a ningún autor en particular.

Así, se puede definir ***desarrollo software*** como *el conjunto de elementos software de diferente nivel de abstracción, producidos en las diferentes fases del ciclo de desarrollo, y que tienen como objetivo común la obtención de un producto software final*.

De forma intencionada se ha omitido de la definición el concepto de aplicación ejecutable, al entender que desarrollar productos que no constituyen por sí una aplicación ejecutable (*framework, bibliotecas de clases o de funciones...*), constituye un esfuerzo de desarrollo software perfectamente válido, y del cual se puede derivar un mecano como elemento reutilizable complejo.

2.2 Elemento software reutilizable (asset)

En el presente trabajo se está definiendo un elemento software reutilizable complejo, definido en varios niveles de abstracción. Pero, este elemento software reutilizable puede verse como un asset de mayor granularidad[3] que acoge en su seno un conjunto de assets de menor granularidad los cuales se encuentran clasificados en una serie de niveles de abstracción y relacionados entre sí. Según esto, se puede definir elemento software reutilizable (*asset) como aquel elemento software, que con independencia de su nivel de abstracción, está diseñado para su uso en desarrollos software diferentes al desarrollo software de origen*.

De la definición anterior cabe recalcar dos aspectos importantes:

- *Se mantiene la extensión del alcance de la reutilización a todas las fases de un desarrollo software.*
- *Se incide en el hecho de que la reutilización requiere de un coste adicional en los proyectos software, coste que se refleja en el esfuerzo de diseño de los diferentes assets para su uso en otros proyectos software.*

Los diferentes assets normalmente comparten una serie de características que influyen activamente en la potenciación de su reutilización [Cybulsky, 1995]:

- *Expresivos*: Los assets están clasificados en un nivel de abstracción adecuado o expresan una utilidad general que los hace susceptibles de ser reutilizados en diferentes contextos y ser aplicados en una amplia variedad de áreas de aplicación.
- *Definidos*: Están construidos y documentados con un claro propósito, sus características y limitaciones son fácilmente identificables, sus interfaces, dependencias externas y entornos de operación están especificados, y cualquier otro requisito existente se encuentra perfecta y explícitamente definido.
- *Transferible*: Es posible transferir el asset a un entorno o dominio de problema diferente al que en origen fue creado. Esto significa que el elemento debe ser lo más independiente posible, con pocas dependencias de implementación, abstracto y bien parametrizado.
- *Aditivo*: Esto es, que sea posible integrarlo con otros assets para formar elementos reutilizables compuestos sin tener que realizar excesivas modificaciones y sin conllevar efectos laterales adversos.

[3] Salvando las distancias se podía comparar la estructura compleja de reutilización con el concepto de mecanismo de organización que introducen los diversos lenguajes de modelado propios de diferentes metodologías de desarrollo orientado a objetos como pueden ser los mecanismos de Booch [Booch, 1994], los clusters de Meyer [Meyer, 1994] o los paquetes de UML [Rational et al., 1997b].

- *Formal:* Los assets debieran poder ser descritos a algún nivel de abstracción mediante una notación formal o semi-formal, de forma que pudiera existir algún mecanismo para poder verificar su corrección, predecir la violación de integridad de sus restricciones a la hora de integrarlo con otros assets o asegurar el nivel de compleción de un producto software construido con assets.
- *Informáticamente representables:* Aquellos elementos software reutilizables que pueden ser descritos en términos de unos valores de unos determinados atributos computacionales, que pueden ser fácilmente descompuestos en partes representables por un ordenador, que pueden ser accedidos, analizados, manipulados y posiblemente modificados por procesos basados en ordenador, tienen un claro potencial para formar parte de una biblioteca flexible de elementos reutilizables. Estos assets pueden ser fácilmente buscados, recuperados, interpretados, modificados y finalmente integrados en sistemas software de mayor entidad.
- *Autocontenidos:* Los assets que encierran una única idea son más fáciles de entender, tienen menos dependencias con factores externos (ya sean de entorno o de implementación), tienen unas interfaces fáciles de utilizar, son fáciles de extender, adaptar y mantener.
- *Independientes del lenguaje:* Los elementos software reutilizables deben omitir los detalles de implementación propios de los lenguajes de programación. Esto es, los elementos software reutilizables debieran ser descritos en términos de formalismos de especificación, o de forma que las soluciones de bajo nivel pudieran ser utilizadas por una amplia variedad de lenguajes de programación sobre una plataforma de implementación dada.
- *Capaces de representar datos y comportamiento:* Deben ser capaces de encapsular sus estructuras de datos y su lógica hasta un grano fino de detalles de forma que se aumente la cohesión y se reduzca el acoplamiento por dependencia de datos comunes.
- *Verificables:* Los assets deben ser fáciles de probar por los encargados de su mantenimiento, y, lo que es más importante, por los usuarios que los utilicen en sus desarrollos con reutilización.
- *Simples:* Las interfaces pequeñas y sencillas ayudan a la reutilización de los assets, así como a su comprensión.
- *Fáciles de cambiar:* Ciertos tipos de problemas requieren assets que adopten nuevas especificaciones sin que ello provoque una gran cantidad de efectos laterales.

Una vez determinado qué es un asset y cuáles son sus características deseables, se debe definir cuál será su estructura estática, es decir, qué campos de información se asociarán a cada elemento componente de la estructura de reutilización compleja. La mejor manera de establecer estos campos de información es realizar un repaso por una serie de repositorios y modelos de elementos reutilizables, de forma que se obtenga un estado del arte sobre la estructura de un asset, en la cual basar la definición que aquí se está exponiendo.

2.2.1 Estado del arte de la estructura de un asset

Como punto de partida de este repaso por las estructuras atómicas de reutilización se toma el proyecto EEC-SPRIT II ITHACA (Integrated Toolkit for Highly Advanced Computers Applications) [Ader et al., 1990]. Este proyecto se llevó a cabo entre 1989 y

1992 con el objetivo de establecer un entorno de desarrollo software soportado en dos bases fundamentales: la orientación a objeto y la reutilización.

El entorno almacena y publica para su reutilización los objetos semánticos, los objetos de diseño y los objetos de implementación dentro del SIB (*Software Information Base*) [Constantopoulos et al., 1992].

El SIB consiste en un conjunto de objetos que representan información del software en diferentes niveles de abstracción (*requisitos, diseño y código*) organizada en descripciones. La estructura de cada descripción depende del modelo de descripción utilizado en cada caso. Un modelo de descripción es un conjunto de metaclases que representan entidades, las relaciones entre dichas entidades y la semántica específica asociada a la forma en que sus instancias se interrelacionan.

El SIB establece una red semántica con las descripciones software. Cada descripción es un nodo en la red; y los enlaces representan las dependencias, correspondencias, relaciones semánticas, transformaciones y enlaces hipermedia a documentación [Bellinzona et al., 1993].

En resumen, de ITHACA se puede destacar:

- Es una referencia obligada para cualquier trabajo de reutilización en varios niveles de abstracción.
- No ofrece una definición clara de lo que sería la estructura de un componente software reutilizable.
- Para la representación de los diferentes tipos de assets utiliza diferentes formalismos y herramientas: F-ORM (Functionality in the Objects with Roles Model) para los requisitos, Visual ADL (Activity Definition Language) para el diseño detallado, Telos para las descripciones almacenadas en el SIB.
- El punto más importante de ITHACA viene de la mano de las relaciones entre componentes.

REBOOT (***REuse Based on Object-Oriented Tecniques***) [Karlsson, 1995], [SER, 1996] es un proyecto europeo **ESPRIT III #7808,** desarrollado dentro del SER ESPRIT Project #9809. El principal objetivo de este proyecto es crear un marco metodológico y organizador para implantar la reutilización como un método habitual en las organizaciones en las que se desarrolla software. Además del enfoque metodológico, REBOOT aporta un modelo de componente software reutilizable, que sirve como marco de referencia para la implementación práctica de algunos repositorios. Para hacer factible la reutilización, los assets deben estar almacenados en el repositorio junto con la información necesaria que permita su recuperación, la evaluación de su ajuste a los requisitos buscados, y facilite su adaptación e integración en el sistema global si fuera necesario. El modelo de componente software reutilizable tiene como objetivo describir la información que se necesita almacenar junto al asset. Este modelo se refleja en la Figura 1, utilizando la notación UML 1.1 [Rational et al., 1997a].

La ***clasificación*** de un asset es la información que se guarda junto con el asset y que sirve de ayuda para la identificación y recuperación del mismo. Es la información en que se basa el desarrollador con reutilización para buscar un asset. La relación entre la clasificación y el componente es de uno a varios. Esto significa que varios componentes pueden tener la misma clasificación.

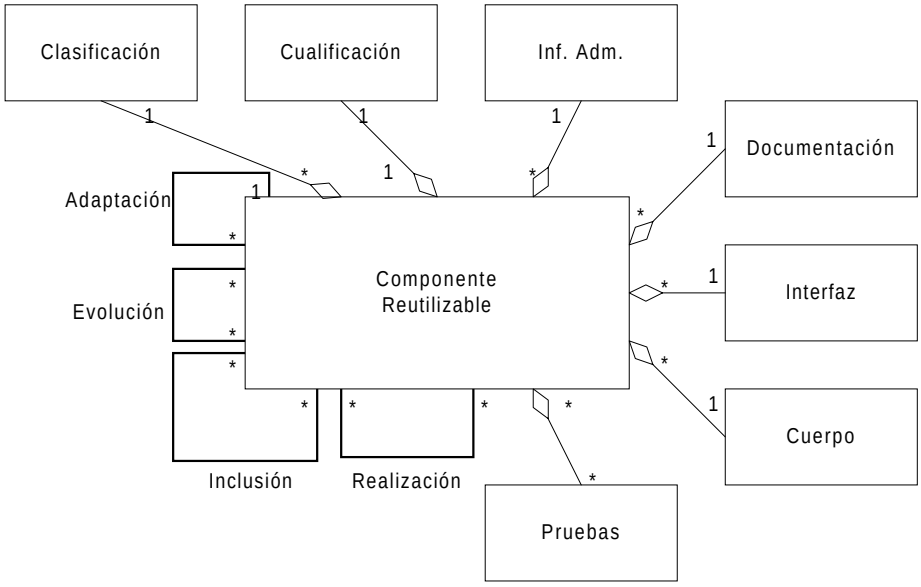

Figura 1. Modelo de componente reutilizable de REBOOT

La ***información de cualificación*** intenta describir la calidad y la capacidad de reutilización del asset en función de una serie de criterios (*portabilidad, adaptabilidad, confidencia…*) y de una serie de métricas. También pueden tener cabida comentarios, problemas surgidos al reutilizar el componente, soluciones a dichos problemas… de forma que se cuenta con una especie de histórico de la reutilización del componente. A cada asset le corresponde su propia cualificación.

La ***información administrativa*** hace referencia a la información general del asset[4], a los niveles de autorización y al coste asociado al uso del asset. La relación entre la información administrativa y el asset es de uno a varios, esto es, una misma información administrativa puede estar conectada a varios componentes, pero cada componente sólo puede tener una información administrativa.

La ***documentación*** es esencial en la reutilización de un componente. Puede ser de dos tipos: *la documentación que facilita la reutilización del asset* y *la documentación del asset que formará parte de la documentación del producto en el que el asset se incluirá.* La relación entre la documentación y el componente es uno a varios.

La ***interfaz*** del asset describe la forma de comunicarse con el componente, es decir, que operaciones ofrece, que parámetros toma y que requiere de su entorno. Por su parte, el ***cuerpo*** del asset es la descripción de como trabaja el componente internamente. Se tiene, por tanto, que REBOOT distingue entre interfaz e implementación, al más puro estilo orientado a objetos. La relación entre la interfaz y el asset es de una a varios, e igualmente sucede con el cuerpo y el asset.

Los ***juegos de pruebas*** de los assets constituyen una información de suma importancia para los assets, ya que éstos deben ser probados con anterioridad a su introducción en el repositorio y volverán a ser probados a la hora de ser reutilizados. La relación entre el componente y las pruebas es varios a varios, ya que un componente puede tener varios juegos de prueba, y varios juegos de prueba se pueden aplicar a varios componentes.

La ***relación de realización*** relaciona assets a diferente nivel de abstracción. De esta forma, si se tiene que reutilizar una especificación, se podría acceder al diseño e

[4] Nombre, dirección, teléfono, correo electrónico... de los desarroladores y encargados del mantenimiento del asset, así como la fecha de inserción en el repositorio y la fecha de última modificación del asset.

implementación correspondientes, ahorrando el trabajo en las fases subsiguientes. La relación de realización es una relación varios a varios entre assets, debido a que una especificación puede estar asociada a muchas soluciones de bajo nivel, y viceversa.

La ***relación de inclusión*** se usa para reflejar la agregación, esto es, asset puede estar formado por varios assets. Los assets componentes pueden reutilizarse también de forma separada, ya que aparecen como unidades independientes en el repositorio. La relación de inclusión es varios a varios, ya que un asset puede ser un agregado de varios assets, pero un asset puede formar parte de varios assets agregados.

La ***relación de evolución*** enlaza las diferentes versiones del mismo asset, mostrando así su histórico de versiones.

La ***relación de adaptación*** muestra las diferentes adaptaciones que ha sufrido un componente para su reutilización en sistemas para los que no cumplía los requisitos completamente. Es una relación uno a varios entre los componentes reutilizables.

Una vez que se ha realizado este repaso por el modelo de asset propuesto por REBOOT, y a modo de resumen, se podrían citar las siguientes conclusiones:

- El objetivo de REBOOT no es ofrecer una definición concreta de los campos de información que deben acompañar al asset en el repositorio.
- Intenta establecer un marco de referencia del tipo de información que debe guardarse de cada uno de los assets, que posteriormente se concretizará en la implementación de los repositorios que tomen a REBOOT como referencia.
- Basado en REBOOT existe la implementación de un repositorio comercial, EUROWARE [SER, 1996].
- El modelo de asset de REBOOT ha ejercido una notable influencia en la primera propuesta de mecano [García et al., 1997a].
- Se presenta una relación entre assets clasificados en niveles de abstracción diferentes, *la relación de realización*. No obstante, la definición que se hace de esta relación es demasiado general, carente de cualquier connotación semántica, convirtiéndola en un simple enlace entre assets.
- Es especialmente pobre en cuanto al modelado de las relaciones semánticas entre los assets clasificados en un mismo nivel de abstracción, contemplado exclusivamente la agregación y el versionado de assets.
- El modelo de asset propuesto por REBOOT abusa de las cardinalidades múltiples entre el asset y los componentes de información. Por motivos de claridad, se debería buscar relaciones uno a uno entre el asset y los componentes de información existentes.
- Algunos de los componentes de información propuestos por REBOOT podían ser considerados como assets reutilizables por sí mismos y asociados con otros assets, por ejemplo el caso de las pruebas y de la documentación.

EUROWARE (***Enabling Users to Reuse Over Wide AREas***) [SER, 1996], [Sema Group, 1996], [Villa, 1997] es el proyecto ESPRIT #8947, desarrollado dentro del proyecto SER ESPRIT #9809. EUROWARE es un conjunto de aplicaciones, construidas sobre la base de REBOOT, e integradas en un servidor WWW que permite que clientes remotos, conectados a una red TCP/IP, tengan acceso a los assets almacenados en el repositorio para su reutilización.

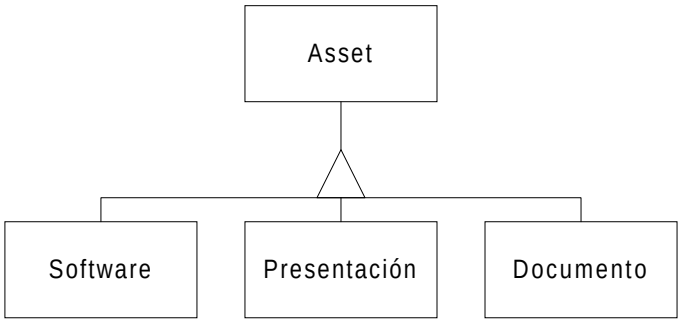

Figura 2. Jerarquía de assets en EUROWARE.

El modelo de datos de EUROWARE presenta una serie de entidades de las cuales, para el fin que se persigue en el presente trabajo, sólo se presentará la entidad **Asset**.

Asset es una clase abstracta que se especializa en tres tipos de assets (*software*, *presentación y documento*), y que recoge los elementos comunes a los subtipos de asset.

Un **asset** tendrá los siguientes atributos:

- **Identificador:** Un identificador de asset único generado por el servidor.
- **Nombre:** Un nombre común para identificar al asset.
- **Descripción:** Un texto de introducción general del asset.
- **Visibilidad de grupo:** Grupos cuyos miembros pueden leer la información del asset, pero no pueden extraerlo del repositorio.
- **Extracción de grupo:** Grupos cuyos miembros pueden extraer el asset del repositorio.
- **Sector de actividad:** Sector de actividad de la organización que creó el asset.
- **Dominio de aplicación:** El dominio de aplicación con el que está relacionado el asset.
- **Tecnología:** Se refiere al campo tecnológico desde un punto de vista de reutilización horizontal (*cliente-servidor*, *interfaz de usuario*...).
- **Tamaño:** El tamaño en KB de los ficheros que componen el asset.
- **Número de versión:** El número de versión del asset.
- **Fecha de creación:** La fecha en que el asset fue generado.
- **Fecha de introducción:** La fecha en que el asset fue insertado en EUROWARE. Este atributo es asignado por el servidor.
- **Fecha de última modificación:** La fecha de última modificación del asset.

Un **asset software** hereda todos los atributos de **asset** y, además, añade los siguientes:

- **Entorno:** Programa que se usó para generar el asset.

- **Contenido:** Indica el tipo de elementos que están incluidos en el asset.
- **Sistema operativo:** El sistema operativo que soporta el asset.
- **Lenguaje:** Si es un código fuente, indicaría el lenguaje de programación en que el asset está escrito.
- **Plataforma:** Describe la plataforma de explotación del asset, incluyendo información sobre el lenguaje fuente/destino utilizado y el sistema operativo.
- **Compilador:** Compilador necesario para compilar el asset.
- **Red:** Tipo de red que solicita el asset.
- **Sistema Gestor de BD**
- **Operaciones**

Un **asset de presentación** hereda todos los atributos de **asset** y, además, añade los siguientes:

- **Formato:** Formato del fichero de presentación.
- **Número de diapositivas**
- **Tipo de presentación:** Comercial, técnica...
- **Script asociado**
- **Referencias:** Referencias a los contenidos de la presentación.

Un **asset documento** hereda todos los atributos de **asset** y, además, añade los siguientes:

- **Formato**
- **Nivel técnico**
- **Audiencia**

Otra de las entidades soportadas por EUROWARE es **Relación**. En este sistema se entiende por relación un enlace bidireccional entre dos entidades. El formato de cada una de estas relaciones es:

Nombre1 (Min1, Max1), Nombre2 (Min2, Max2), T1, T2

Donde:

- **Nombre1** representa el enlace directo entre una instancia de **T1** y una instancia de **T2**.
- **Nombre2** representa el enlace inverso, que enlaza la misma instancia de **T2** con la misma instancia de **T1**.
- **Min1** y **Max1** representan el número mínimo y máximo, respectivamente, de enlaces del tipo **Nombre1** que pueden existir entre una instancia de **T1** y una instancia de **T2**. Recíprocamente, **Min2** y **Max2** representan el número mínimo y máximo, respectivamente, de enlaces del tipo **Nombre2** que pueden existir entre una instancia de **T2** y una instancia de **T1**.

Existen varias relaciones en EUROWARE, pero las que directamente afectan al objetivo del presente documento son:

- *Is-composed-of (0: n), part-of (0: n), Asset, Asset*

  Representa la agregación de assets. Esto es, un asset agregado de varios assets.

- *Uses (0: n), is-used-by (0: n), Asset, Asset*

  Identifica el conjunto de assets independientes que son usados por un asset.

- *Previous-version (0, 1), next-version (0 n), Asset, Asset*

  Identifica la versión anterior de un asset

Asociada a cada asset también se tiene la información sobre la persona que lo insertó. Los campos de información que se almacenan son: **nombre, número de teléfono, número de fax, correo electrónico, dirección de correo postal, nombre de la compañía, teléfono de la organización** y **fax de la organización.**

A modo de resumen, las conclusiones que se sacan del estudio realizado sobre el modelo de datos de EUROWARE, son las siguientes:

- EUROWARE implementa un subconjunto de las propuestas realizadas en el modelo de asset de REBOOT.
- El servidor ofrece cierto grado de flexibilidad al permitir cambiar la definición de las entidades del repositorio.
- La idea de considerar una jerarquía de assets es un elemento positivo. Por una parte establece qué elementos básicos va a tener todo asset, pero permite que cada tipo concreto de asset establezca cuáles serán sus atributos intrínsecos que lo definen y lo diferencian de los assets pertenecientes a otras categorías.
- Sin embargo, la jerarquía que establece EUROWARE es demasiado simplista, y quizás se pueda catalogar de algo artificiosa. Por una parte, el tipo software es algo demasiado genérico que requeriría de una mayor especialización, y por otro lado el tipo presentación parece un apéndice que se puede eliminar, ya que su funcionalidad podía ser asumida por el tipo documento.
- La información de cualificación está presente en EUROWARE, pero pasa bastante desapercibida.
- Las relaciones entre assets son simples enlaces, cuyo único contenido semántico es el que transmite el nombre del enlace. No obstante, debe hacerse hincapié en que este tipo de solución se haya muy extendida en las implementaciones de repositorios que admiten relaciones entre assets. Este mecanismo puede utilizarse para generar un mecano reutilizables estático en el que los enlaces careciesen de contenido semántico.

El repositorio **RIB** (*Repository In a Box*) de **NHSE** (*National HPCC Software Exchange*), es un conjunto de herramientas para la creación de repositorios software que puedan compartir información a través de Internet. RIB presenta una extensión del **BIDM** (*Basic Interoperability Data Model*) que es el estándar de IEEE 1420.1 [IEEE, 1995], para la catalogación de software en Internet. El BIDM ha sido desarrollado por el **RIG** (*Reuse Library Interoperability Group*) [NHSE, 1997a], [NHSE, 1997b].

El BIDM es un modelo objeto que consta de clases que tienen atributos y se relacionan con otras clases. Las cuatro clases de BIDM son ***Asset, Elemento, Biblioteca*** y ***Organización***. Un asset es una entidad reutilizable generada en el ciclo de vida de desarrollo. Un elemento es un fichero. Una biblioteca está compuesta de assets. Y por último, una organización puede ser una persona, una compañía, un grupo de investigación... que crea y gestiona un asset o una biblioteca [NHSE, 1997b].

La extensión de BIDM que presenta RIB es **ACF** (*Asset Certification Framework*). Esta extensión incluye clases adicionales sobre la información de certificación del asset, o lo que es lo mismo, para la revisión o evaluación del asset.

En [NHSE, 1997b] se presenta el modelo de datos de BIDM utilizando la notación de OMT [Rumbaugh et al., 1991], aunque aquí se va a presentar, sin pérdida alguna de información o semántica, utilizando UML 1.1 [Rational et al., 1997a], véase Figura 3. Además, en [NHSE, 1997b] se incluye un glosario explicando todos los términos de BIDM y ACF.

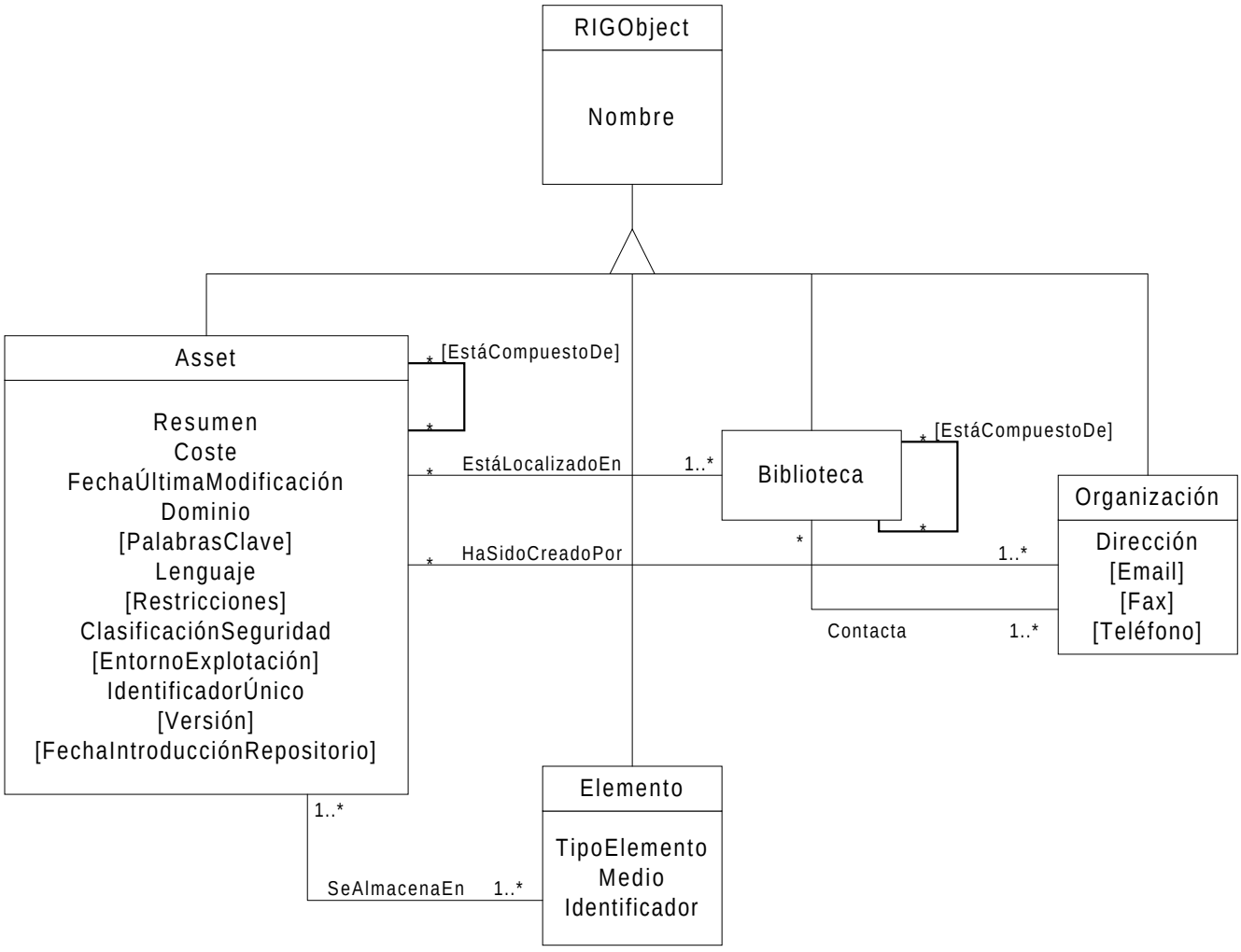

Figura 3. Basic Interoperatibility Data Model

Del estudio de RIB se pueden derivar las siguientes conclusiones:

- RIB utiliza una extensión de BIDM, que es un estándar de modelo de datos propuesto por IEEE.
- Para la definición del modelo de datos subyacente se emplea un modelo objeto, en el que se tienen cuatro objetos principales: Asset, Biblioteca, Elemento y Organización. Estos cuatro objetos se derivan de un objeto raíz denominado RIBObject.
- La definición de los atributos del objeto Asset constituye una referencia perfectamente válida a seguir.
- El ACF de RIB es una apuesta clara por introducir temas de calidad en los assets.
- Su punto más flojo es la relación entre assets, contemplando sólo la relación de composición.

RBSE (Repository Based Software Engineering) [Eichmann, 1995] es un proyecto de investigación desarrollado en el Research Institute for Computing and Information Systems de la Universidad de Houston, y que tiene como objetivo crear un mecanismo de transferencia de tecnología para mejorar las capacidades en ingeniería del software de la NASA, dando soporte a la reutilización del software mediante un repositorio.

RBSE tiene dos ramas principales: la iniciativa de *ingeniería de reutilización* y la *iniciativa de repositorio*. La parte de ingeniería de reutilización se basa en dos pilares que son el ISC (Information Systems Contract) y el proyecto ROSE (Reusable Object Software Engineering). La parte de repositorio descansa sobre MORE (Multimedia Oriented Repository Environment) que ofrece todo un sistema de gestión de repositorio. Como caso concreto o instancia de MORE se tiene a ELSA (Electronic Library Services and Applications) que expande la vista tradicional de biblioteca software con la adquisición, clasificación, almacenamiento y mantenimiento de todo tipo de assets, con la potencia añadida que otorga la hipermedia dentro de un entorno WWW. La arquitectura de RBSE queda reflejada en la Figura 4.

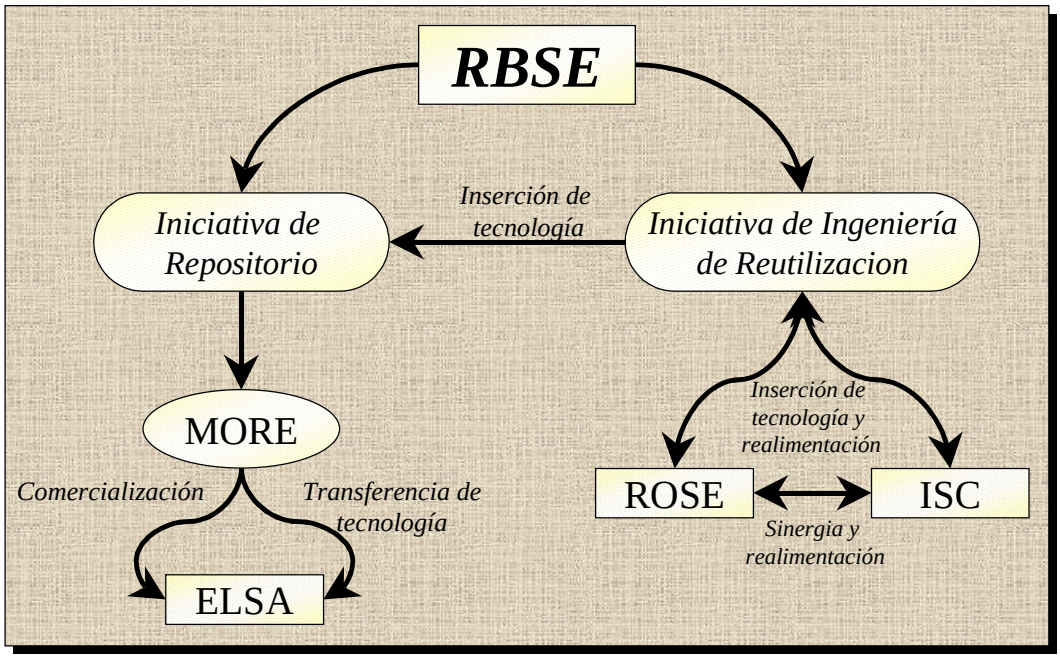

Figura 4. Arquitectura de RBSE.

La construcción de MORE produjo un rediseño de los repositorios anteriores para cubrir los siguientes objetivos:

- *Mecanismo flexible de definición de grupos para gestionar el acceso a las subcolecciones de assets.*
- *Interfaz de usuario basado en WWW.*
- *Integración de otras fuentes de Internet en la interfaz de MORE mediante el uso de URL en el repositorio de datos.*

Para la representación de los assets ELSA se basa en el estándar propuesto por el RIG (*Reuse Library Interoperability Group*) a IEEE, esto es, en BIDM [Eichmann et al., 1995].

A través de MORE, ELSA presenta los assets en una jerarquía de colecciones y clases. De acuerdo a su tema, los assets se incluyen en aquellas colecciones que mejor representen el dominio en el que están definidos. Los assets también se clasifican atendiendo a la clase o el tipo de información. La clase define los atributos de los assets pertenecientes a dicha clase. Así, una superclase incluiría todos los atributos comunes a todas sus subclases, por ejemplo Identificador de asset, Nombre y Tipo de nodo, de esta forma todos los assets de tipo superclase tendrían esos atributos, y todas las clases descendientes de superclase los heredarían, incluyendo éstas los atributos que las definen específicamente [Eichmann et al., 1995].

Como resumen de las características de RBSE, o más concretamente de MORE/ELSA, se pueden citar:

- RBSE supone un marco de mejora para la reutilización en el que se distinguen la parte de ingeniería y la parte de repositorio.
- Como repositorio se define MORE y como ejemplo práctico de MORE se tiene ELSA.
- Se tiene una estructura jerárquica de colecciones para clasificar los assets. Una de las aportaciones más interesantes de estos repositorios es la clasificación que se introduce en las clases, de forma que la superclase define los atributos generales a esa clasificación de assets, y cada tipo de asset específico define además sus propios atributos.
- La estructura de campos de los assets se toma de BIDM, añadiendo un campo que relaciona el asset con la colección o colecciones a las que pertenece.
- El punto más flojo de este repositorio es el tocante a las relaciones entre asset, ya que se limita a definir la relación de especialización para construir las jerarquías de assets.

A la hora de llevar la reutilización a la práctica surgen una serie de problemas operativos, apareciendo diferentes aproximaciones para darles solución, hasta aquí se han estudiado algunas de las diferentes soluciones que se han ido dando. El desarrollo de estos proyectos ha producido un soporte metodológico junto con sus correspondientes herramientas para el almacenamiento, clasificación y recuperación de assets utilizando la red Internet y un servidor World Wide Web. La utilización de la flexibilidad y el poder que ofrece la tecnología web puede aprovecharse para crear un entorno que dé soporte al desarrollo con reutilización y para la reutilización de forma distribuida a través de Internet [Trump, 1997].

Existen actualmente una serie de repositorios que se basan en ficheros HTML (HyperText Markup Language) entre los que cabría citar a ASSET, PAL, CARDS, COSMIC, DSRS y STARS[5]. Realizar un estudio de cada uno de estos repositorios se cree un trabajo innecesario, máxime cuando ya se han estudiado otros importantes repositorios y modelos, pero para terminar este repaso se va a realizar un somero repaso a ASSET por ser el más representativo de este tipo de repositorios.

ASSET (Asset Source for Software Engineering Technology) [SAIC, 1997] es una división de SAIC (Science Applications International Corporation), que ofrece una forma de comercio electrónico basado en WWW.

Dentro de ASSET se encuentra la biblioteca WSRD[6] (Worldwide Software Resource Discovery) que es una biblioteca de software libre y comercial, que está en continuo crecimiento, y que está centrada en artefactos del ciclo de vida del software, así como en documentos sobre desarrollo y reutilización.

Los atributos (*o metadatos como los denomina esta biblioteca*) que tiene un asset particular serían los siguientes:

- **Identificador único:** Identificador único dentro de la biblioteca WSRD.
- **Nombre del asset:** Nombre o título del asset.
- **Versión:** Identificación de la versión.
- **Fecha de creación:** Fecha en que fue creado el asset.
- **Nombre del creador:** Nombre de quien creó el asset.
- **Nombre de la organización:** Organización que creó el asset.
- **Número de referencia:** Un identificador externo a SAIC/ASSET.
- **URL(s):** Localización WWW del asset o de información relacionada con el asset.
- **Tamaño:** Tamaño en KB, número de ficheros o páginas de un documento.
- **Entorno(s):** Entornos destinos, tales como sistema operativo, plataforma…
- **Lenguaje(s):** Si es un código fuente, el lenguaje(s) en que está escrito.
- **Distribución:** Quién puede acceder al asset.
- **Formato(s):** Medios físicos (cinta, vídeo…) o software (pdf, word…) que se requieren para almacenar al asset.
- **Nivel de evaluación:** Indicador de calidad propio de SAIC/ASSET[7].
- **Evaluación del encargado del repositorio:** Texto de explicación de las evaluaciones a las que se ha sometido al asset.
- **Idioma:** Idioma de la documentación.
- **Fecha de entrada:** Fecha de entrada del asset en WSRD.
- **Tipo de asset:** Software, Documento o Referencia.
- **Colección:** Dónde se ha originado el asset.
- **Dominio:** Área de aplicación del asset.

[5] EUROWARE y ELSA entrarían entre los repositorios que basan su interfaz de usuario en WWW, pero además se ven complementados por otras tecnologías que los hacen más completos que los aquí mencionados que se basan exclusivamente en páginas HTML.

[6] Que se pronuncia "*wizard*".

[7] Un asset es evaluado por el encargado del repositorio y se le asocia uno de los siguientes cuatro niveles: **Nivel 1: Documentado, Nivel 2: Auditado, Nivel 3: Compilado** y **Nivel 4: Validado**.

- **Función:** Qué hace el asset.
- **Objeto:** Objeto pasivo de la acción del asset.
- **Palabras clave:** Palabras clave, frases que describen aspectos importantes del asset.

Como resumen de lo visto sobre WSRD de SAIC/ASSET, se puede decir que:

- Repositorio de estructura muy simple, que recoge campos de información de los assets sin establecer jerarquías de ningún tipo.
- Las relaciones entre assets no existen. Las relaciones de composición se logran aumentando el nivel de granularidad del asset, esto es, haciendo que en un fichero comprimido se encuentren todos los ficheros requeridos por el asset.

2.3 Nivel de abstracción

En el momento en que el ámbito de la reutilización se amplía para dar cobertura a elementos diferentes del código fuente, se necesita establecer algún criterio para proceder a la clasificación de los assets.

En la literatura se tienen numerosos ejemplos de criterios de clasificación de assets. Entre estas clasificaciones, se puede señalar la propuesta por Charles W. Krueger [Krueger, 1992], que define una taxonomía de la información reutilizable basándose en niveles de abstracción. Cada nivel de abstracción establece el tipo de asset que se reutiliza y las abstracciones que se usan para describir estos assets. Utilizando este criterio obtiene ocho categorías de reutilización, que van desde los niveles más bajos de abstracción hasta los niveles más altos. Estas categorías son: *lenguajes de alto nivel, aprovechamiento de código y diseño, componentes de código fuente, esquemas software, generadores de aplicación, lenguajes de muy alto nivel, sistemas de transformación* y *arquitecturas software*.

Por su parte, Mili et al. [Mili et al., 1995] parten de una visión de transformación de los sistemas para establecer su taxonomía de enfoques de reutilización. Esta se puede resumir en: *componentes de código fuente, esquemas software, sistemas de transformación reutilizables* y *lenguajes de muy alto nivel.*

Otra categoría es la que establece T. Casper Jones en [Jones, 1984], por la que identifica cuatro tipos de componentes reutilizables: *datos reutilizables, arquitecturas reutilizables, diseños reutilizables* y *programas reutilizables.*

En [Edwards et al., 1997] se propone una clasificación de los componentes relacionados con el proceso de la Ingeniería del Software en dos dimensiones ortogonales. La primera dimensión estaría formada por componentes abstractos (*especificaciones*) y por componentes concretos (*implementaciones*). La segunda dimensión está compuesta por componentes plantillas y por componentes instancias.

En [García et al., 1997a] se presenta un criterio de clasificación basado en una 3-tupla <fase, abstracción, genericidad> donde cada uno de los elementos de la tupla representa una dimensión ortogonal al resto, obteniendo un espacio tridimensional en el que clasificar los assets.

Sin embargo, la forma más habitual de encuadrar los assets en la literatura es atendiendo a una clasificación muy simple según la fase del desarrollo y/o el nivel de abstracción en el que el conocimiento se produce o se reutiliza. Este doble criterio establece la

reutilización según tres niveles: *reutilización de código, reutilización de diseños* y *reutilización de especificaciones*[8] [Constantopoulos et al., 1992], [Bellinzona et al., 1993], [Karlsson, 1995], [McClure, 1997].

Tomando como referencia la clasificación de los assets según la fase del desarrollo, se puede definir un nivel de abstracción de un asset como *la etapa del ciclo vital de desarrollo software caracterizada por la cercanía al nivel del dominio del problema o al nivel del dominio de la solución de los elementos software generados en dicha etapa.*

2.4 Relaciones dentro de un mismo nivel de abstracción

Las relaciones entre los assets clasificados en un mismo nivel de abstracción es un concepto de fundamental importancia en la reutilización del software.

Si se tuviera que justificar la importancia que para la reutilización del software tienen las relaciones entre los assets de un mismo nivel se podría afirmar que, con independencia del paradigma de desarrollo, no suele servir de nada la reutilización aislada de un asset sino se acompaña de todos los assets con los cuales esté relacionado. Esta aseveración es especialmente importante en los desarrollos orientados al objeto, donde, por ejemplo, es sumamente extraño que un programador reutilice una clase en solitario.

Esta afirmación se ve refrendada por tres principios básicos del diseño orientado a objetos [García et al., 1997b], como son ***el principio de equivalencia reutilización/revisión*** [Martin, 1996], ***el principio de cierre común*** [Martin, 1996] y ***el principio de reutilización común*** [Martin,1996].

Así, el principio de ***equivalencia reutilización/revisión*** se enuncia: *La granularidad de la reutilización es la granularidad de la revisión. Sólo los componentes que son revisados a través de un sistema de distribución pueden ser reutilizados de forma efectiva. Este grano es el paquete.*

Por su parte, el principio de ***cierre común*** dice que: *Las clases en un paquete deben estar cerradas juntas frente a los mismos tipos de cambios. Un cambio que afecta a un paquete afecta a todas las clases del paquete.*

Por último, el principio de ***reutilización común*** afirma que*: Las clases que pertenecen a un paquete se reutilizan juntas. Si se reutiliza una de las clases del paquete, se reutilizan todas.*

Con los tres principios anteriores queda más que demostrada la importancia para la reutilización de las relaciones entre los diferentes assets de un mismo nivel. Además, en el repaso realizado de los modelos de asset en el apartado ***2.2.1 Estado del arte de la estructura de un asset***, se pudo apreciar que las relaciones entre assets aparecían en la mayoría de estos modelos, aunque estas relaciones se limitaban a ser simples enlaces entre assets, relegando las connotaciones semánticas al nombre dado a cada tipo de relación soportada por el modelo.

El siguiente punto a considerar es la identificación de las relaciones estructurales que establezcan los tipos de relaciones semánticas que existirán entre assets clasificados en un mismo nivel de abstracción.

[8] Un claro ejemplo de esta clasificación se encuentra en el proyecto ITHACA [Constantopoulos et al., 1992], [Bellinzona et al., 1993]

En el proyecto ITHACA se tenía que en el SIB se almacenaban las descripciones de los objetos software y sus relaciones semánticas. Las relaciones semánticas del modelo ITHACA se clasifican en tres categorías:

- ***Relaciones semánticas estructurales generales:*** Son los mecanismos básicos de modelado que ofrece el lenguaje de representación del conocimiento. A esta categoría pertenecen: la propiedad de tener atributos, la clasificación y la generalización.
- ***Relaciones semánticas estructurales especiales:*** Tienen la misión de definir un conjunto minimal de descriptores especiales. Estarían incluidas la agregación, la correspondencia, la similitud y la especifidad.
- ***Asociaciones:*** Descriptores que permiten la construcción de vistas mediante consultas.

REBOOT presenta tres relaciones entre assets de un mismo nivel de abstracción, la relación de inclusión que indica agregación y dos relaciones para asociar versiones de assets, la relación de evolución y la relación de adaptación.

En EUROWARE aparecen las relaciones como enlaces bidireccionales. De todas las relaciones que presenta EUROWARE, las que relacionan assets entre sí indican agregación, uso y relaciones entre versiones de assets.

Por su parte, RIB sólo contempla la relación de composición, mientras que RBSE de forma explícita no soporta relaciones entre assets, aunque de forma implícita está presente la relación de especialización/generalización.

Como puede apreciarse, a excepción de ITHACA, los modelos estudiados tienen en las relaciones entre assets su asignatura pendiente. Aparece la agregación como la relación estructural más significativa, apareciendo las relaciones de asociación y generalización/especialización de forma implícita en todos ellos, estando el tema del versionado de assets presente en algunos modelos.

Dado que los modelos estudiados son a todas luces insuficientes en el tema de las relaciones estructurales, se va a proceder a realizar un estudio de los diferentes modelos objeto[9] presentes en las principales metodologías de desarrollo orientadas a objeto.

Como referencias representativas se han seleccionado OMT [Rumbaugh et al., 1991], Método Booch [Booch, 1994], UML 1.1 [Rational et al, 1997a] y OML [Firesmith et al., 1996]. Del estudio de estos modelos objeto se pueden obtener las siguientes conclusiones:

- *Cada propuesta define un modelo objeto diferente, con un núcleo común pero con detalles diferenciales de suma importancia. Estos modelos pueden ser simples, aunque semánticamente ricos, como en el caso de OMT o el Método Booch, o por el contrario pueden ser más complejos como OML o UML.*
- *Aunque todos estos modelos objetos presentan relaciones semánticas diversas, se puede decir que hay tres relaciones básicas que todos ellos comparten: la relación de asociación, la relación todo/parte y la relación es-un.*

[9] Se ha centrado el estudio en los modelos objeto exclusivamente tratando de simplificar, al ser los modelos objeto mucho más expresivos y semánticamente más ricos que los modelos estructurados.

- *Estas tres relaciones centrales deben servir como base para obtener un conjunto de relaciones de semántica más especializada.*
- *Las relaciones estructurales deben soportar las relaciones semánticas entre elementos del dominio, que no este caso no es otro que el dominio del desarrollo de software con independencia del paradigma.*

El control de versiones es otro aspecto importante a tener en cuenta en un entorno de reutilización, y por consiguiente las relaciones que se derivan de este hecho. Randy Katz establece las relaciones semánticas propias del control de versiones en bases de datos de diseño: *es parte de, derivación, configuración, equivalencia* y *herencia* [Katz, 1990]

Según lo visto hasta el momento, se puede decir que *dos elementos software, pertenecientes al mismo nivel de abstracción, están relacionados entre sí cuando existe un enlace semántico entre ellos*.

2.5 Relaciones entre assets de niveles de abstracción diferentes

La identificación de las relaciones entre assets pertenecientes a un mismo nivel de abstracción es un proceso imprescindible para la consecución de la reutilización de todos los assets relacionados en dicho nivel. Pero, cuando el objetivo está en extender la reutilización a un ámbito multinivel de abstracción, se necesita de un proceso complementario consistente en definir las posibles relaciones entre assets situados en niveles de abstracción diferentes.

Estas relaciones entre niveles son las que van a sustentar el concepto de la estructura multinivel como elemento software reutilizable definido en varios niveles de abstracción, sirviendo de enlace entre los assets componentes de dicha estructura que se encuentran clasificados en diferentes niveles de abstracción.

El principal objetivo de estas relaciones es el conseguir que una vez seleccionado un determinado asset, definido en un nivel de abstracción cualquiera, se puedan reutilizar tanto los assets relacionados con él en el mismo nivel de abstracción, como los assets relacionados en otros niveles de abstracción diferentes.

De esta forma, se puede decir que *dos assets pertenecientes a diferentes niveles de abstracción están relacionados entre sí cuando el elemento de menor abstracción se ha generado mediante un proceso de refinamiento o transformación del elemento de mayor nivel de abstracción, o cuando el elemento de mayor nivel de abstracción se ha generado como resultado de un proceso de ingeniería inversa a partir de un elemento de abstracción menor*.

De la definición se puede sacar la primera característica o restricción de este tipo de relaciones: ***son relaciones binarias***.

Tradicionalmente, la reutilización se ha dirigido exclusivamente en un solo nivel de abstracción, típicamente el nivel de implementación, aunque como ya se ha discutido en este mismo documento existen numerosos trabajos que abogan por extender la reutilización a niveles mayores de abstracción. Pero, no existen demasiados trabajos que intenten relacionar el mismo elemento software en diferentes niveles de abstracción, con lo que el número de referencias que sirvan de guía a la hora de definir las relaciones multinivel, no van a ser tan numerosas como en el caso de relacionar elementos en un mismo nivel de abstracción.

Como primera referencia obligada se tiene de nuevo al proyecto ESPRIT ITHACA. Este proyecto tenía como objetivo ofrecer un entorno y un método para construir nuevas aplicaciones reutilizando assets de antiguas aplicaciones [Bellinzona et al., 1993]. Los aspectos metodológicos de ITHACA se basan en la reutilización; la reutilización del software se ve reflejada en su sistema de repositorio, llamado SIB (*Software Information Base*). Este repositorio almacena assets de diferentes niveles de abstracción, concretamente descripciones de requisitos, descripciones de diseño y descripciones de implementación [Ader et al., 1990]. Esto significa que especificaciones software de más alto nivel pueden ser reutilizadas directamente, pero también pueden servir como índices a assets definidos en niveles de abstracción menores.

Si se habla de ITHACA como ejemplo de reutilización multinivel, es obligado introducir su modelo de reutilización. El tipo de asociación más importante dentro del SIB, y con una granularidad gruesa, es el marco de aplicación (*Application Frame*).

Los marcos de aplicación representan sistemas completos o familias de sistemas que tienen al menos una implementación y opcionalmente descripciones de diseño y de requisitos. Los marcos de aplicación pueden catalogarse en genéricos y específicos.

Los assets se agrupan en el SIB como GAFs (*Generic Application Frames*). Un GAF es una abstracción de una colección de aplicaciones de un dominio específico de aplicación, que incluye una descripción de requisitos, una o más descripciones de diseño y una o más descripciones de implementación para cada descripción de diseño. Las clases reutilizables que representan los requisitos, el diseño detallado y la implementación se encuentran relacionadas en un GAF, convirtiéndose éstas en los nodos del grafo y los enlaces entre ellos son las relaciones semánticas que relacionan los elementos reutilizables. Además, los GAFs pueden incluir sugerencias de cómo reutilizar las clases que contiene.

Por su parte, un SAF (*Specific Application Frame*) describe un sistema completo e incluye exactamente una implementación.

Para conseguir que los GAFs sean reutilizables, éstos deben ser diseñados de la forma más genérica posible, y los enlaces que relacionan los diferentes niveles de abstracción (*especificación, diseño* e *implementación*) deben ser apropiadamente diseñados y ofrecer una buena información.

REBOOT en su modelo de asset presenta la relación de realización como un enlace entre assets clasificados en diferentes niveles de abstracción.

Incidiendo en la idea de reutilización multinivel, se tiene otro interesante concepto en el ***kit específico de un dominio***, que puede definirse como *un conjunto de productos de trabajo (elementos software producidos en diferentes puntos del ciclo de vida del software) compatibles y reutilizables, que son construidos para trabajar bien juntos y diseñados para construir fácilmente un conjunto relacionado de aplicaciones* [Gris, 1993]. El primer paso para juntar varios assets de un dominio específico compatibles es identificar y estructurar el dominio (*ingeniería del dominio*) y después utilizar métodos (*ingeniería del kit*) para construir los kits.

Una evolución del concepto de kit específico de dominio es el concepto de kit híbrido específico de un dominio [Griss and Wentzel, 1995a]. Un kit híbrido está compuesto de frameworks específicos de un dominio, assets, un lenguaje de configuración y una serie de herramientas de soporte [Griss and Wentzel, 1995b].

Los kits híbridos están entre las bibliotecas de componentes y otros elementos más altamente configurables o generadores de aplicaciones. El concepto de kit híbrido presentado por Griss y Wentzel es perfectamente válido tanto para la construcción de software bajo el paradigma objetual, como para la construcción de aplicaciones tradicionales. En [Griss and Kessler, 1996] se introduce de nuevo el concepto de kit específico de un dominio desde el prisma de la orientación a objeto.

Ciñéndose al tema concreto de este apartado, esto es, a las relaciones entre niveles de abstracción diferentes, existen dos relaciones semánticas muy utilizadas en inteligencia artificial y en la definición de arquitecturas de modelado en orientación a objetos, que relacionan conceptos definidos en diferentes niveles de percepción. Estas relaciones son la ***reificación*** y la ***reflexión***.

La palabra *reificación* es un vocablo que no existe en español, ni tampoco es un concepto propio de la informática, por lo tanto antes de entrar en definiciones concretas se va a hacer una introducción desde el punto de vista etimológico de la palabra. Reificación es un anglicismo o traducción al español del término inglés *reification*, que se deriva del verbo *to reify*. Este verbo que tiene su origen en el latín "**re[s]**" que significa "*cosa*" y en el sufijo inglés "**ify**" que literalmente significa tratar una abstracción como si fuera un objeto real.

Así, si se consultan algunos diccionarios se pueden obtener definiciones de reificación como las que siguen: "*El proceso de considerar algo abstracto en una entidad material*" [Web, 1997], "*Considerar algo abstracto como una cosa material o concreta*" [Webster, 1996].

En general, se podía definir la reificación como *el proceso de hacer que un concepto proveniente de un nivel de abstracción superior o inferior sea visible en el nivel de percepción actual* [Peuker, 1997] o como *el proceso de hacer que conceptos externos estén disponibles en el nivel objeto, y el uso de dichos objetos reificados es lo que se conoce como reflexión* [Madany et al., 1991].

En el campo de la inteligencia artificial suele utilizarse el término reificación en el sentido del *proceso por el que una expresión se convierte en un objeto (un valor) de un lenguaje particular* [Plaza, 1997]. Un sentido acorde a los intereses del presente trabajo es el que a la reificación le da Cyrano [Haase, 1996], donde la reificación es el proceso por el cual comportamientos de un nivel de dominio se hacen visibles en objetos de otro nivel.

Por su parte, en el campo de la orientación a objetos, la reificación aparece en diferentes contextos. Ejemplos representativos de la reificación en la orientación a objetos pueden ser: la reificación de atributos que se produce cuando se convierte un atributo de una clase en una clase, esto es, se ha reificado un atributo para convertirlo en un tipo [Johannsson et al., 1996], la arquitectura de cuatro capas[10] (*o jerarquía de modelos*) donde el metamodelado se basa en la idea de reificar las entidades que forman un cierto tipo de modelo y describir las propiedades comunes del tipo de modelo en forma de un modelo de objetos [Rivas et al., 1997], o el mecanismo clásico para obtener la reflexión.

Así, la reificación puede verse como la acción por la cual existe una transferencia de información desde un mecanismo interno de un sistema procedural, representacional o racional, a un dominio en el que se puede proceder, representar o razonar, es decir,

[10] Es el framework conceptual de metamodelado generalmente aceptado. Explica las relaciones entre el meta metamodelo, el metamodelo, el modelo y el nivel de datos de usuario. Juntos forman las cuatro capas una encima de la otra [Metamodel, 1997].

convertir algo interno en una "*cosa*". Mientras que la reflexión es la acción por la cual la información es transferida desde un dominio a la parte interna de un sistema, esto es, la internalización de la información.

Sin duda alguna, la referencia más adecuada para justificar la consideración de la reificación como la relación estructural entre assets clasificados en diferentes niveles de abstracción, se encuentra en los trabajos que sobre reificación se han realizado en el departamento de bases de datos de la Universidad de Braunschweig (Alemania) [Denker and Ehrich, 1995], [Denker, 1995], [Huhn et al., 1995], [Denker, 1996], [Huhn et al., 1996]. En dichos trabajos se presenta a la reificación como una acción de refinamiento por la cual se reduce la complejidad del proceso de diseño software. Visto desde un prisma objetual, una especificación encapsula estructura y comportamiento, debido a lo cual se distingue entre reificación de datos y reificación de acción.

El uso de la palabra reificación en lugar de los vocablos refinamiento o implementación enfatiza el cambio de granularidad [Denker and Ehrich, 1995]. Se pretende expresar el diferente punto de vista con el que se puede mirar un elemento software, de forma que es atómico desde un punto de vista y compuesto desde otro. La reificación significa que desde la especificación inicial a la implementación se construyen una secuencia de especificaciones, de forma que en cada paso la especificación que se logra está más cercana a un programa concreto.

3. Mecanos reutilizables

En el apartado anterior se han establecido las bases sobre las que se puede definir una estructura de reutilización compleja. Ahora, tomando dichas pautas como referencias, y haciendo uso de la experiencia adquirida en proyectos sobre reutilización software [Villa, 1997], [Martínez y Maudes, 1997], se va a definir la estructura compleja de reutilización que de soporte a un elemento software reutilizable, de forma que cumpla los siguientes principios [García et al., 1998]:

- ***Aumento del nivel de abstracción de la reutilización:*** Debe aumentarse el alcance de la reutilización del software, dirigiéndolo hacia niveles de mayor abstracción que la implementación.
- ***Estructura multinivel de abstracción:*** La estructura debe acoger varios elementos software reutilizables relacionados entre sí y clasificados en diferentes niveles de abstracción.
- ***Definición de un soporte para el desarrollo para reutilización:*** La estructura debe dar soporte a los elementos reutilizables que se hayan generado en un esfuerzo de desarrollo software.
- ***Base para el desarrollo con reutilización:*** La estructura debe ofrecer una serie de facilidades que ofrezcan la flexibilidad necesaria para su reutilización en desarrollos futuros.

La estructura definida bajo las perspectivas anteriormente expuestas va a recibir el nombre de **mecano**, pudiéndose definir como: "*Un sistema de elementos software reutilizables, clasificados en diferentes niveles y relacionados entre sí, ya sea dentro de un mismo nivel de abstracción (relaciones intranivel) o entre diferentes niveles de abstracción (relaciones internivel), cumpliéndose la restricción de que debe existir al menos una relación internivel*".

Los mecanos se pueden dividir en dos categorías *mecanos de geometría estática (mecanos estáticos)* y *mecanos de geometría variable.*

Un mecano estático se produce cuando se establece que un determinado sistema de assets se encuentra listo para ser almacenado en un repositorio formando un elemento software reutilizable con una geometría estática, que no variará en el tiempo por acciones diferentes a las propias del mantenimiento del repositorio. De aquí se deduce que los mecanos estáticos se encuentran vinculados al desarrollo para la reutilización, ya sea desde su concepción inicial o como resultado de un proceso previo de desarrollo con reutilización con mecanos estáticos (*de forma que un mecano de geometría variable se convierte en un mecano estático*).

Un mecano de geometría variable se produce en tiempo de reutilización, cuando para un desarrollo con reutilización se accede al repositorio y se recupera un conjunto de elementos software reutilizables relacionados entre sí, pero que no tienen que corresponderse con un mecano estático, ni pertenecer todos a la misma estructura estática.

3.1 Requisitos de los mecanos de geometría estática

Como paso previo al estudio detallado de los apartados que conforman las bases para establecer el modelo básico de mecano, se van a enumerar una serie de requisitos que debe cumplir esta estructura compleja de reutilización [García et al., 1998].

Los primeros requisitos se derivan directamente de la propia definición de mecano:

- ***Los assets componentes de un mecano deben estar clasificados en un determinado nivel de abstracción.*** Siendo los niveles de abstracción en los que se clasifican los assets son tres: *especificación, diseño e implementación.* Los niveles *conceptual, lógico y físico* han sido considerados tradicionalmente como los niveles de abstracción en la producción de software. Sin embargo, buscando un paralelismo con la fase del ciclo de vida en que el asset ha sido generado se ha preferido utilizar como niveles de abstracción el nivel de especificación, el nivel de diseño y el nivel de implementación

- ***Se tienen dos tipos de relaciones entre assets: relaciones intranivel y relaciones internivel.*** Teniendo en cuenta que los assets están clasificados en un determinado nivel de abstracción, se cuenta con dos tipos de relaciones entre assets; las relaciones entre los assets pertenecientes a un mismo nivel de abstracción y las relaciones entre assets pertenecientes a diferentes niveles de abstracción, denominándose relaciones intranivel y relaciones internivel respectivamente.

- ***En todo mecano siempre hay más de un nivel de abstracción representado.*** Se ha defendido en numerosos trabajos que el aumento del nivel de abstracción de los elementos software reutilizables provocan un incremento de los beneficios potenciales de la reutilización. No obstante, aunque el aumento del ámbito de la reutilización es positivo, se sigue presentando a los assets definidos en un solo nivel de abstracción o correspondiéndose a una sola fase del ciclo de vida de desarrollo del software. Como consecuencia de esto, ni los desarrolladores para reutilización, ni los desarrolladores con reutilización tienen concepción de la reutilización sistemática que puede afectar a varios niveles de abstracción al mismo tiempo. Para extender el concepto de reutilización hacia una visión de alcance multinivel, se define una estructura compleja de reutilización que dé origen a un elemento software reutilizable definido en diferentes niveles de abstracción.

Este mismo requisito lleva a la situación de cualquier mecano estático debe contar siempre con al menos una relación internivel. Esta situación da lugar a cuatro posibles configuraciones:

1. **E-D-I**: Sería el caso normal. Se tienen assets definidos en los tres niveles de abstracción (*especificación, diseño e implementación*) y relaciones internivel entre ellos.
2. **E-D**: Un caso probable en el que se tienen especificaciones y diseños relacionados por relaciones internivel.
3. **E-I**: Se tienen assets en los niveles de especificación e implementación, pero no en el nivel de diseño.
4. **D-I**: Otro caso que, aunque no es el más interesante, es muy frecuente. Se tienen assets en todos los niveles de abstracción a excepción de en el nivel de especificación.

Sin embargo, existen otra serie de requisitos que se derivan de un estudio más detallado y de la experiencia en el proceso de desarrollo para reutilización, a saber:

- ***Todo asset debe ser de un tipo predefinido, que indique de qué clase de artefacto software se trata.***
- ***Se debe tener una flexibilidad para incrementar los tipos de assets soportados.***
- ***La estructura del asset debe dar soporte a la información lógica y física del asset, además de ofrecer detalles sobre la seguridad, la calidad y los aspectos administrativos del mismo.***
- ***Un asset puede formar parte de varios mecanos estáticos.*** Un mismo elemento software reutilizable puede formar parte de varios esfuerzos de desarrollos (*eso al menos es el objetivo de la reutilización*), y por lo tanto puede aparecer como componente de distintos mecanos estáticos que conviven en el repositorio.
- ***Las relaciones semánticas (relaciones del dominio) entre los assets se derivan del dominio del desarrollo del software.*** No debe perderse de vista que al definir una estructura de elemento software reutilizable, se está inmerso en el dominio del desarrollo de software, y por tanto no debe extrañar el hecho de que muchas de las relaciones semánticas coincidan en nombre con las relaciones estructurales necesarias para establecer el modelo.
- ***Las relaciones del dominio deben definir perfectamente las restricciones necesarias para evitar la introducción de incongruencias en la estructura de reutilización.*** Más concretamente, las relaciones semánticas deben establecer las restricciones oportunas para evitar incongruencias a la hora de enlazar tipos de assets, incorporando de esta forma una semántica que se echaba en falta en los modelos de elementos software reutilizables existentes.
- ***Un mecano estático puede estar formado por un conjunto de mecanos estáticos reutilizables estáticos existentes.*** Como se ha expresado anteriormente, un mecano reutilizable estático es un agregado de assets relacionados. Pero, también puede definirse en función de otros mecanos reutilizables estáticos.
- ***Un mecano estático está ligado a un dominio.*** La reutilización del software es una de las prácticas que mayores beneficios en productividad y calidad puede ofrecer en el campo del desarrollo de software. Pero, cuando la reutilización se conduce hacia

una visión multinivel de la misma, se está caminando hacia una reutilización vertical en dominios definidos y precisos.

- ***Un mecano estático está ligado a un contexto de reutilización.*** El contexto de un mecano reutilizable estático es el entorno en el que se le permite operar. Contiene las restricciones que han de cumplirse en el entorno en el que el mecano será reutilizado.

- ***Un mecano estático debe soportar la evolución de sus assets componentes.*** La modificación de un asset es una operación básica que debe darse en cualquier repositorio. Según como se realice esta modificación, se tendrá una modificación de un asset o una versión del mismo.

  Cuando el proceso de modificación se realiza con bloqueo, implica que mientras se está editando nadie más lo puede leer/editar. De las modificaciones que se efectúen no saldrá una nueva versión, sino una modificación de la versión en cuestión que sustituye a la que hubiera antes. Este proceso es típico de una operación de mantenimiento del mecano reutilizable estático por parte del responsable de la gestión del repositorio, siendo parte de las responsabilidades de los procesos de desarrollo para reutilización.

  Cuando el proceso de modificación se realiza sin bloqueo, se está trabajando con una copia del asset, por lo que el asset puede seguir siendo manipulado por el resto de usuarios mientras se está modificando éste en algún espacio privado de trabajo. Este es un proceso típico de un desarrollo con reutilización, cuando se necesita adaptar alguno de los assets reutilizados. Si se decide incorporar el componente modificado en el repositorio, se pueden dar los siguientes casos:

  - Se decide sustituir el asset del repositorio por el adaptado. En este caso no se generaría versión, sino una modificación que afectaría a todos los mecanos de los que formara parte el asset modificado.

  - Se introduce el asset adaptado en el repositorio, pero sin sustituir el asset que ya estaba, generándose una versión de éste. La versión de asset de la que procede otro asset se denomina ancestro, simétricamente el nuevo asset es un descendiente de su ancestro. Esta relación ancestro – descendiente se conoce por relación de derivación. Los assets por tanto describen un historial de versiones a través de la relación de derivación, que será un árbol si se restringe a uno el número de ancestros de una versión.

    Esta situación provoca que todos los mecanos reutilizables estáticos que contuvieran a dicho asset se verían afectados con la existencia de la nueva versión de su componente.

  - El tercer caso posible, es que se introduzca al repositorio un nuevo mecano que comparta los assets no modificados con los mecanos ya existentes, e incorporé los assets adaptados como nuevos assets al repositorio, no como versiones de assets ya existentes.

4. Mecanos estáticos

Teniendo en cuenta la definición de lo que es un mecano, así como los requisitos que debe cumplir esta estructura compleja de reutilización, se va a proceder a establecer lo que sería el modelo de la estructura estática de reutilización que se ha bautizado con el nombre de mecano.

Pero, para seguir un paralelismo con el estudio previo realizado, se van ir fijando cuales son los aspectos concretos en cada una de las bases establecidas para la definición de un mecano estático: *el nivel de abstracción, la estructura del asset, las relaciones entre assets de un mismo nivel de abstracción y las relaciones entre assets de diferentes niveles de abstracción.*

4.1 Niveles de abstracción en un mecano reutilizable estático

Buscando una definición acorde con las fases genéricas de un desarrollo software, aquí se va a utilizar el criterio más extendido de asociar el nivel de abstracción a la fase de generación del asset.

Esta solución tiene la ventaja de ser intuitiva y el inconveniente de ser demasiado pobre de cara a la selección de assets. Esta limitación se puede solventar utilizando este criterio exclusivamente para marcar los estados de abstracción que comprende un mecano reutilizable estático. Esto obliga a definir otro criterio de clasificación basado en facetas de cara a la selección y recuperación de los assets, en el cual el nivel de abstracción o fase de generación sea una de las facetas a considerar, pero no la única.

Según los expresado anteriormente, los mecanos estáticos estarán compuestos por tres niveles de abstracción, que en orden decreciente de abstracción serían **nivel de requisitos**[11], **nivel de diseño** y **nivel de implementación**.

4.2 Estructura del asset GIRO

La estructura de los assets que componen un mecano estático está basada en el modelo BIDM [IEEE, 1995], utilizándose para su representación un modelo objeto.

El centro de este modelo será la clase **Asset**, pero deben existir otras entidades con las que se relaciona la clase **Asset**, estas entidades están representadas por las clases **Representación** y **Creador**. En la Figura 5 se puede apreciar un primer modelo del núcleo básico de lo que será el modelo de asset GIRO[12].

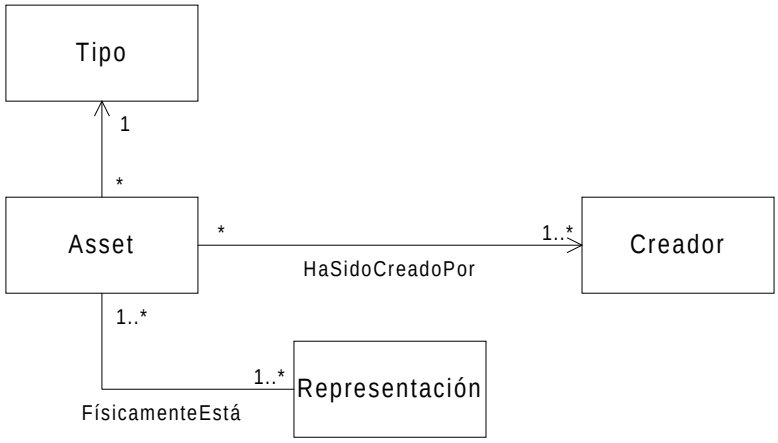

Figura 5. Modelo básico de asset.

[11] Otra alternativa válida, la cual se ajustaría perfectamente a los objetivos que se están marcando, hubiera sido considerar cuatro niveles en lugar de tres, es decir, el nivel de requisitos podría verse dividido en dos niveles: ***nivel de definición de requisitos*** y ***nivel de especificación de requisitos***. Pero, finalmente se ha optado por la división en tres niveles por motivos de sencillez, debido a la existencia de assets que carecen de un criterio preciso para determinar a cual de esos dos primeros niveles pertenecería.

[12] Para la realización de los modelos va a utilizarse la notación UML 1.1 [Rational et al., 1997c].

El siguiente paso es el refinamiento de este modelo[13], encontrando cuales son los elementos estructurales (*atributos*) y los métodos de cada una de las clases.

La clase **Asset** representa un asset lógico genérico, esto significa que sus atributos van a recoger la información necesaria para la definición del asset, pero sin reflejar ninguna connotación física del mismo. Presenta los siguientes los siguientes atributos:

- ***Identificador:*** Es un identificador único dentro del sistema.
- ***Nombre:*** Es el nombre o el título.
- ***Resumen:*** Explicación o definición general del asset.
- ***FechaCreación:*** Fecha de creación del asset.
- ***FechaIntroducción:*** Fecha en que el asset fue introducido en el repositorio.
- ***FechaModificación:*** Fecha de última modificación del asset.
- ***Dominio:*** Área de aplicación, actividad o conocimiento del asset.
- ***PalabrasClave:*** Palabras o frases que describen al asset.
- ***NivelAbstracción:*** Fase del ciclo de vida en que fue generado el asset. Para ser consecuentes con la faceta del nivel de abstracción, este atributo puede tomar uno de los valores siguientes: **requisitos, diseño** o **implementación.**
- ***Método:*** Método al cual pertenece el asset.
- ***Paradigma:*** Paradigma de desarrollo bajo el que se ha generado.
- ***Entorno:*** Entorno de destino del asset, esto es, plataforma hardware, sistema operativo y/o compilador necesarios para que el asset sea operativo.
- ***Restricciones:*** Información legal sobre el uso del asset, incluyendo copyright, patentes, derechos de explotación, limitaciones de exportación y licencias.
- ***NivelSeguridad:*** El nivel de seguridad más alto asignado a un asset o cualquier parte constituyente de un asset.
- ***Coste:*** Cuantía que debe pagarse por la reutilización del asset.
- ***Idioma:*** Lengua en la que se encuentra la documentación del asset.
- ***Versión:*** Designación de la versión de un asset.
- ***NivelEvaluación:*** Nivel de evaluación del asset establecido por el responsable del repositorio. Se establecen cinco posibles valores: **desconocido** (*el asset no ha sido revisado*), **revisado** (*el responsable del repositorio ha comprobado que el asset se encuentra completo*), **auditado** (*el responsable del repositorio ha confirmado que están los componentes que debían estar y su estado de integridad*), **compilado** (*los programas fuentes han sido compilados y/o los documentos han sido revisados para detectar deficiencias o inconsistencias internas*) y **validado** (*el responsable del repositorio ha seguido el protocolo de certificación impuesto al repositorio para validar el asset*).

Además, todo asset tiene un tipo de asset que se representa por la clase **TipoAsset.**

[13] Siguiendo los pasos básicos del AOO.

En cuanto a los métodos de la clase **Asset**, a parte de aquellos métodos que permiten la modificación o consulta de los atributos, destacar la necesidad de establecer los mecanismos de creación adecuados para los assets.

Los assets pueden estar relacionados entre sí de diferentes formas. No obstante, esta característica será ampliada en los subapartados 4.3 y 4.4.

La clase **Representación** determina la parte física de un asset, esto es, toda la información relacionada con la localización física del asset. Los atributos que presenta dicha clase son:

- ***Identificador:*** Es un identificador único dentro del sistema.
- ***Nombre:*** Es el nombre o el título.
- ***URL:*** Uniform Resource Locator del fichero en el que se encuentra el asset, esto permite hacer referencia a assets distribuidos en una Intranet o en Internet.
- ***Formato:*** Formato del fichero (ASCII, TeX, Postscript, binario…).
- ***Tamaño:*** Tamaño del asset (KB, páginas…).
- ***Medio:*** El medio en el que puede obtenerse el asset (CD-ROM, papel, vídeo…).
- ***Comentarios:*** Posibles notas sobre el formato físico del asset.

La clase **Creador** representa la organización o el autor(es) responsable(s) de la creación y mantenimiento del asset. Los atributos de esta clase son:

- ***Identificador:*** Es un identificador único dentro del sistema.
- ***Nombre:*** Es el nombre o el título.
- ***Razón Social:*** NIF o CIF del responsable.
- ***Dirección:*** Dirección postal.
- ***Email:*** Correo electrónico.
- ***Teléfono:*** Número de teléfono.
- ***Fax:*** Número de fax.
- ***Comentarios:*** Comentarios de relevancia sobre el creador del asset.

La clase **Asset** está relacionada con la clase **Creador** mediante la asociación ***HaSidoCreadoPor,*** de forma que todo asset tenga asociado al menos un responsable, ya sea una organización o una persona concreta. La clase **Asset** se encuentra relacionada con la clase **Representación** mediante la asociación ***FísicamenteEstá*** de forma que cualquier asset tiene que estar localizado al menos en un lugar físico determinado, aunque es posible que se encuentre en más de un lugar y, por otra parte, un determinado continente físico, por ejemplo un fichero, puede contener más de un asset lógico.

Con lo visto hasta el momento, el modelo básico definido en la Figura 5 podría refinarse y completarse de la siguiente forma:

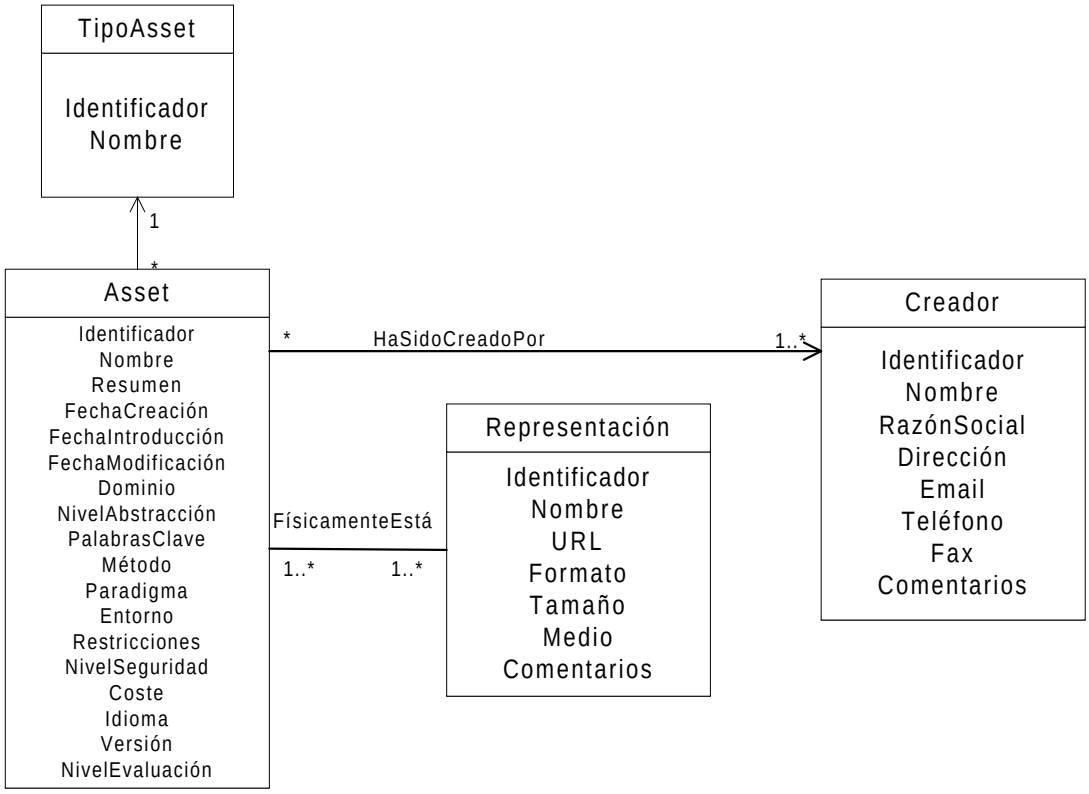

Figura 6. Modelo básico de asset GIRO.

4.3 Relaciones entre assets GIRO definidos en un mismo nivel de abstracción

Las relaciones entre assets tienen un papel protagonista dentro de un entorno de reutilización sistemática. Al hablar de relaciones entre assets se está haciendo referencia a las relaciones semánticas que se dan entre los objetos del universo de discurso en el que se plantea la definición de los mecanos estáticos.

Teniendo en cuenta que los assets cuentan con un atributo que caracteriza su nivel de abstracción, se ha optado por la diferenciación de las relaciones semánticas entre los assets pertenecientes a un mismo nivel de abstracción y las relaciones semánticas entre assets pertenecientes a diferentes niveles de abstracción, denominándose ***relaciones intranivel*** y ***relaciones internivel*** respectivamente.

Aunque el presente apartado se centra en el estudio de las relaciones entre assets del mismo nivel de abstracción, se puede definir la primera característica común a ambos tipos de relaciones: *las relaciones entre assets van a ser binarias*. Esto es así porque las relaciones semánticas binarias son más sencillas de modelar y de entender, y porque cualquier otra relación de mayor orden puede representarse mediante un conjunto finito de relaciones binarias. El uso de relaciones binarias para representar relaciones entre elementos de modelado es una práctica común de la que se tienen destacados ejemplos en el modelo BRM (*Binary Relations Model*) [Abrial, 1974], [Bracchi et al., 1976], [Girow, 1996] y en el estándar ODMG 2.0 [Cattell and Barry, 1997].

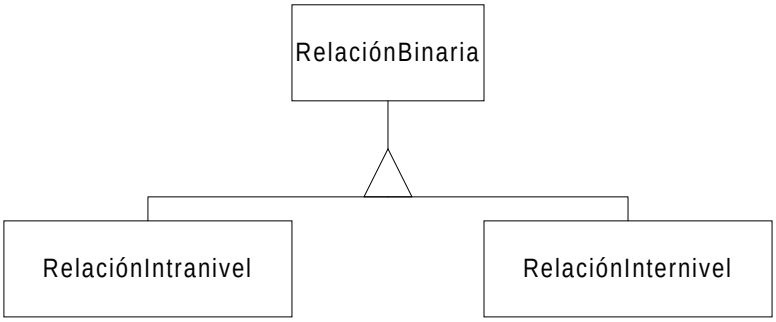

Figura 7. Tipos de relaciones entre assets.

La clase **RelaciónBinaria** es una clase abstracta que recoge una definición general de lo que es una relación binaria. Desde un punto de vista estático una relación binaria entre dos asset es un enlace entre ellos que conlleva una semántica asociada, y que queda definido mediante los siguientes atributos:

- ***Identificador:*** Identificador único de la relación.
- ***Nombre:*** Nombre que se le da a dicha relación.
- ***IdAsset1:*** Identificador del asset desde el cual parte la relación.
- ***IdAsset2:*** Identificador del asset que es destino de la relación.

La clase **RelaciónIntraNivel** es una subclase de **RelaciónBinaria** que incluye la restricción **Asset1.Fase = Asset2.Fase,** mientras que la clase **RelaciónInterNivel** es una subclase de **RelaciónBinaria** que incluye la restricción **Asset1.Fase ≠ Asset2.Fase**.

Una vez hecha esta introducción, va a procederse a la identificación de los tipos de relaciones estructurales que tienen que estar soportadas por el modelo objeto subyacente que de lugar al modelo de mecano, y por tanto a las relaciones semánticas intranivel entre assets necesarias que influyen en su reutilización.

En las relaciones intranivel se tienen relaciones de definición y relaciones de enlace [Firesmith et al., 1996]. Las relaciones de definición son relaciones binarias entre assets, donde un asset depende de la definición ofrecida por el otro, mientras que las relaciones de enlace, como su propio nombre indica, son enlaces semánticos entre assets.

Dentro de las relaciones de definición, y desde el punto de vista de la reutilización, se requiere dar soporte a una relación imprescindible, la relación es-un. La relación de definición por excelencia es la relación de taxonomía, herencia o relación generalización/especialización, que se da entre un elemento más general y un elemento más específico. Sin entrar en discusiones sobre la herencia, ni en clasificaciones sobre la misma[14], se puede afirmar que la relación de herencia se corresponde con la relación básica “es-un”.

De una forma más precisa, se puede definir la relación de herencia como *la relación de definición entre un hijo y uno de sus padres, donde el hijo es una nueva definición creada a partir de una o más definiciones existentes y ofrecidas por los padres* [Firesmith et al., 1996].

[14] Para una completa clasificación de los tipos de herencia se recomienda la consulta del capítulo 24 “*Using inheritance well*” de la segunda edición del libro clásico de Meyer [Meyer, 1997].

Por lo que respecta a las relaciones de enlace en reutilización, se va a trabajar con un conjunto de relaciones derivadas de la asociación, las cuales van añadiendo semántica a la generalidad exhibida por la relación de asociación.

Las asociaciones son las relaciones más genéricas que existen, es decir, las relaciones que menos contenido semántico aportan, resultando extremadamente útiles para el enlace de assets.

La relación de asociación es una relación bidireccional por definición, por lo tanto los dos asset relacionadas mediante una asociación deben saber el uno del otro. Sin embargo, en muchas ocasiones sólo interesa que sea uno de los assets el que tenga conocimiento del asset que está al otro lado de la relación, esta es la típica situación en una relación cliente/servidor, donde el cliente es el que tiene que tener conocimiento del servidor y no viceversa, por tanto se está utilizando una asociación unidireccional, que recibe el nombre de relación de uno o usa-a [Booch, 1994].

Con las asociaciones y la relación usa-a se recogen los enlaces más típicos entre dos assets definidos en un mismo nivel. Pero, en algunas ocasiones se necesita una asociación más restrictiva, este tipo de asociación viene representado por la asociación de dependencia [Rational et al., 1997b]. La relación de dependencia se puede definir como *una asociación entre dos assets, de forma que un cambio en uno de los assets (el asset independiente*) afectará al otro asset (*el asset dependiente*)[15].

Otra relación semántica que debe entrar a formar parte de las relaciones entre los assets de un mecano estático es la relación ***todo/parte***, que establece una relación de enlace entre un asset compuesto y sus partes componentes. Esta relación ***todo/parte*** va a aparecer representada en dos relaciones: *la agregación* y *la composición*.

La agregación es un *tipo especial de asociación que especifica una relación todo/parte entre el agregado (todo) y un componente (parte)* [Rational et al., 1997b]. Se puede distinguir un tipo de agregación todavía más restrictiva y que recibe el nombre de composición. La composición puede definirse como *la relación semántica que describe una forma de agregación con un matiz fuerte de propiedad y de coincidencia de vida como partes de un todo. Las partes con una multiplicidad que no es fija pueden ser creadas después del objeto compuesto, pero una vez creadas ellas viven y mueren con él. Estas partes pueden ser también eliminadas antes de la muerte del objeto compuesto* [Rational et al., 1997b].

Tanto la agregación como la composición son relaciones esenciales para la reutilización. Si se tiene que un asset está compuesto de otros elementos, a la hora de reutilizar dicho asset habrá que reutilizar también todos sus componentes. La decisión de utilizar agregación o composición para expresar una relación todo/parte debe ser tomada por el responsable del mantenimiento del repositorio que recoja los mecanos reutilizables estáticos, y dicha decisión se tomará en función de si las partes pueden sobrevivir o no a la destrucción del agregado.

La agregación (*y la composición*) es la relación que va a permitir marcar cual es el grado de granularidad de los assets que componen un mecano reutilizable estático. Por ejemplo, un DFD puede introducirse como un único asset, o puede introducirse como un asset compuesto por una serie diagramas, que a la vez están compuestos de procesos, siendo los procesos primitivos assets también.

[15] Por ejemplo la relación entre el código fuente de un componente OCX y el binario correspondiente.

Una vez establecidas cuales son las relaciones estructurales necesarias, habría que identificar cuales son las relaciones semánticas o relaciones del dominio entre los assets. A la vez que se identifican deben fijarse las restricciones de las mismas para evitar incongruencias en los enlaces entre assets.

La lista definitiva de relaciones intranivel ha de confeccionarse según se vayan estableciendo los tipos de assets. No obstante, volver a hacer hincapié en el hecho de que el dominio que se está modelando con el modelo de mecano estático es el dominio del desarrollo de software, con lo cual en la lista de relaciones semánticas intranivel van a aparecer nombres de relaciones que son a su vez relaciones estructurales (*herencia, agregación, composición, uso, dependencia…*).

Un aspecto importante que debe fijarse es la dirección de las relaciones. La mayoría de las relaciones semánticas serán unidireccionales, esto es, tienen un único sentido definido por la relación. Sin embargo, para que las relaciones semánticas sean útiles deben ser navegables en los dos sentidos, es decir, que si interesa se pueda acceder desde el asset destino de una relación al asset que la origina[16]. Este problema puede ser solucionado de diversas formas. Una de las soluciones posibles podría ser emular a EUROWARE, y a cada relación unidireccional asociarle otra relación unidireccional en sentido inverso[17], pero esto da lugar a una mayor cantidad de relaciones que no aportan nueva semántica.

La solución que aquí se propone es darle un significado de dirección a los atributos que representan los identificadores de los assets en las relaciones, es decir que cada asset que intervenga en una relación asuma un determinado rol, así, **IdAsset1** siempre debe representar el asset que da origen a la relación, e **IdAsset2** siempre debe representar el asset destino de la relación. Este convenio evita duplicar las relaciones en los casos de relaciones unidireccionales. De esta forma la Tabla 1 recoge los roles que deben representar los identificadores de los assets en las diferentes relaciones semánticas identificadas entre assets clasificados en un mismo nivel de abstracción.

Relación	IdAsset1	IdAsset2
Herencia	***Padre***	***Hijo***
Instanciación	***Clase genérica***	***Clase instanciada***
Asociación	***Indiferente***	***Indiferente***
Uso	***Cliente***	***Servidor***
Dependencia	***Dependiente***	***Independiente***
Agregación/Composición	***Todo***	***Parte***

Tabla 1. Ejemplo de los roles que toman los assets en algunas relaciones intranivel.

4.4 Relaciones entre assets GIRO definidos en diferentes niveles de abstracción diferentes

La idea que subyace bajo las relaciones internivel es representar un mismo concepto que ha ido evolucionando a lo largo de diferentes niveles de abstracción, normalmente

[16] Por poner un ejemplo, una agregación va desde del "*todo*" al componente, de forma que si se quiere reutilizar el todo se tenga acceso a las partes. Sin embargo, también debería ser posible que desde las partes se tuviera acceso al todo.

[17] Por ejemplo la relación de agregación llevaría asociada la relación SerAgregadoDe.

mediante un proceso de refinamiento desde un nivel de mayor abstracción a otro nivel más concreto, aunque no puede omitirse el caso contrario, es decir, que un asset de un nivel de abstracción menor sea el origen de uno o varios assets en un nivel de abstracción mayor.

Para representar este tipo de relaciones semánticas entre assets clasificados en diferentes niveles de abstracción se ha elegido la **Reificación**, como ya se justificó anteriormente. Las relaciones semánticas de este tipo serán relaciones binarias unidireccionales entre dos assets clasificados en niveles de abstracción diferentes, donde **IdAsset1** representaría el identificador del asset origen de la relación e **IdAsset2** representaría el identificador del asset destino de la relación.

4.5 Mecano como agregación de mecanos

Se ha definido un mecano estático como un conjunto de assets clasificados en diferentes niveles de abstracción y relacionados entre sí. Se han establecido los niveles de abstracción y la estructura de los assets, así como las relaciones entre ellos.

Pero una vez definida esta estructura, puede plantearse la siguiente posibilidad: un mecano reutilizables estático puede contener a su vez otro mecanos reutilizables estáticos, entrando en la agregación de mecanos reutilizables estáticos.

La estructura definida en la Figura 8 puede permitir esta situación de forma que el mecano resultante fuera la unión de los assets agregados en los mecanos estáticos que entrarían a formar el nuevo mecano estático.

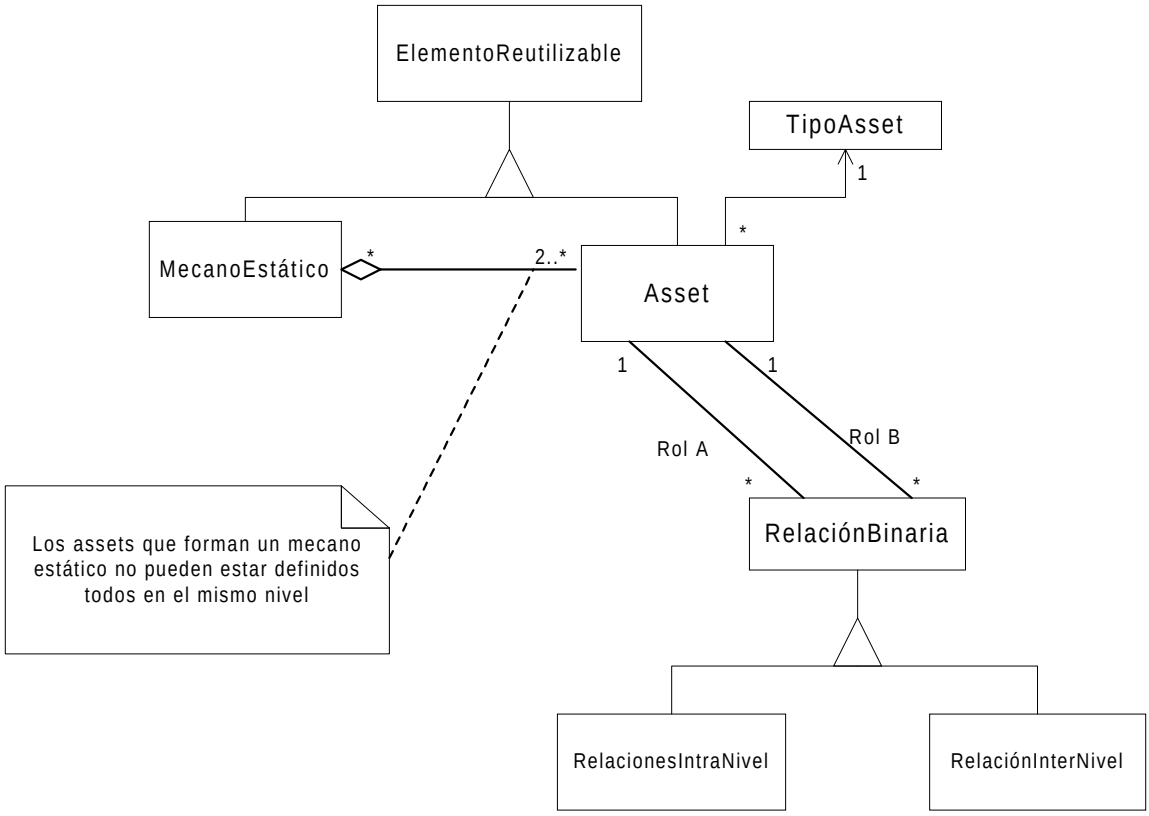

Figura 8. Modelo preliminar de un mecano estático.

Pero esta aproximación tiene un problema, no se tiene definida la operación unión de mecanos reutilizables estáticos.

Se puede optar por otra solución que vendría de la mano de ampliar la definición de un mecano estático para incluir la posibilidad de construcción de mecanos mediante la agregación de éstos, quedando la definición de mecano reutilizable estático como sigue: *Se denomina **mecano estático** a un subconjunto de un desarrollo software compuesto por elementos software reutilizables, provenientes de dicho desarrollo software, clasificados en diferentes niveles de abstracción y relacionados entre sí, ya sea en el mismo nivel de abstracción o en niveles de abstracción diferentes, o bien a aquel mecano constituido como agregación de mecanos reutilizables estáticos ya existentes.*

4.6 Modelo de mecano estático

Después de lo visto en los apartados anteriores se está en condiciones de representar lo que sería el núcleo del modelo de asset GIRO, modelo que puede apreciarse en la Figura 9.

A la vista de este modelo, a modo de conclusiones, deben realizarse los siguientes comentarios:

- *En primer lugar, volver a recalcar el hecho de que el modelo que se ha creado es un modelo que representa la vista estática de la estructura compleja de reutilización que lleva por nombre **mecano estático**.*
- *Se ha utilizado un modelo objeto para representar las entidades semánticas del problema. No obstante este modelo objeto no está completo porque está centrado en las características estáticas (atributos y métodos) de dichas entidades, faltando todo el aspecto dinámico de las mismas.*
- *Se ha añadido una clase semántica a las identificadas inicialmente, la clase **MecanoEstático**, que representa el elemento reutilizable complejo que es una agregación de assets.*
- *Que un mecano estático sea una agregación y no una composición es fácil de argumentar. Cada uno de los componentes de un mecano reutilizable estático puede ser parte de otros mecanos reutilizables estáticos, la eliminación de un mecano reutilizable estático no conlleva obligatoriamente a la eliminación de sus componentes.*
- *De los dos puntos anteriores se puede concluir que de las diferentes connotaciones semánticas que se le pueden asociar a la relación todo/parte, en el caso de los mecanos estáticos la semántica asociada a esta relación es la de la agregación según UML 1.1, y más concretamente, se corresponde con lo que este lenguaje de modelado denomina agregación compartida, y que no es más que un tipo de agregación débil que implica que si se destruye el todo no se tienen que eliminar las partes constituyentes, y lo que es más importante, las partes constituyentes pueden aparecer en varios agregados al mismo tiempo.*
- *En el campo de las relaciones no se han contemplado inicialmente las relaciones de versionado por considerarse estas una consecuencia directa del uso del entorno de reutilización.*

- *Aunque es un resultado más propio del comportamiento dinámico de los mecanos, la estructura estática contempla la posibilidad de que un mecano estático pueda ser una agregación de otros mecanos reutilizables estáticos.*
- *Se ha dado por sentado a lo largo de todo el documento que el repositorio es el soporte necesario para el almacenamiento de los assets producidos como consecuencia de los desarrollos para reutilización siendo, además, el soporte al desarrollo de sistemas soportando reutilización.*
- *El objetivo que se persigue con la definición de esta estructura no es el de construir una herramienta CASE, ni un motor de repositorios, sino que desde la perspectiva de los repositorios, definir un esquema que permita almacenar, manejar y recuperar elementos reutilizables, más concretamente mecanos.*
- *Con el modelo generado se tienen prácticamente las mismas posibilidades que ofrecen los repositorios más utilizados, algunos de los cuales han sido objeto de estudio en el presente trabajo. Incluso en el área de las relaciones semánticas se ha establecido las bases para un modelo más rico, aunque queda por determinar la lista final de relaciones semánticas entre los assets.*
- *Si un mecano se ve como el modelado conceptual de una estructura de información formada por assets y sus relaciones, el mecano estático estaría al mismo nivel que un esquema de bases de datos, siendo el sistema gestor de bases de datos el repositorio. Bajo esta perspectiva, el nivel de datos correspondería a los assets introducidos en el repositorio, el nivel de modelado sería el modelo de mecano estático, habiéndose utilizado como lenguaje de modelado UML 1.1.*
- *La idea presentada en el punto anterior es válida como primera aproximación, pero debe cambiarse en próximas versiones para adoptar un metamodelo que presente un concepto central, la metaclase, de la cual serían instancias todas las clases que representan las entidades del modelo de mecano estático.*
- *Bajo este prisma, la inclusión de nuevos tipos de assets no contemplados en el momento de definición del modelo inicial se llevaría a cabo creando nuevas instancias de la clase* ***TipoAsset****. En cuanto a la inclusión de nuevos tipos de relaciones semánticas se llevaría a cabo extendiendo el modelo derivando las clases* ***RelaciónIntraNivel*** *y/o* ***RelaciónInternivel,*** *obteniéndose una nueva versión del esquema del mecano estático, cumpliendo el principio abierto/cerrado del DOO [Meyer, 1997].*

5. Conclusiones y trabajo futuro

Con este informe se recoge gran parte del trabajo realizado durante el primer año en el seno del grupo GIRO, obteniéndose como principal resultado la definición informal de una estructura compleja de reutilización, la cual se ha denominado mecano, distinguiéndose entre mecano de geometría estática y mecano de geometría variable.

Del estudio detallado del proceso de desarrollo para la reutilización se ha generado un modelo que representa el núcleo de la estructura de los mecanos estáticos, dejándose unas bases sentadas para la creación de un entorno de reutilización sistemática.

Por hacer quedan muchas tareas, en primer lugar terminar de modelar algunos aspectos de los mecanos estáticos como son la identificación de los tipos de assets y de las relaciones semánticas entre ellos, la definición de una política de certificación de assets y los temas relacionados con el versionado de assets.

Con este trabajo se debe empezar a trabajar en el estudio del proceso de desarrollo con reutilización para finalmente establecer un marco formal que aúne ambos procesos en un marco único de reutilización sistemática.

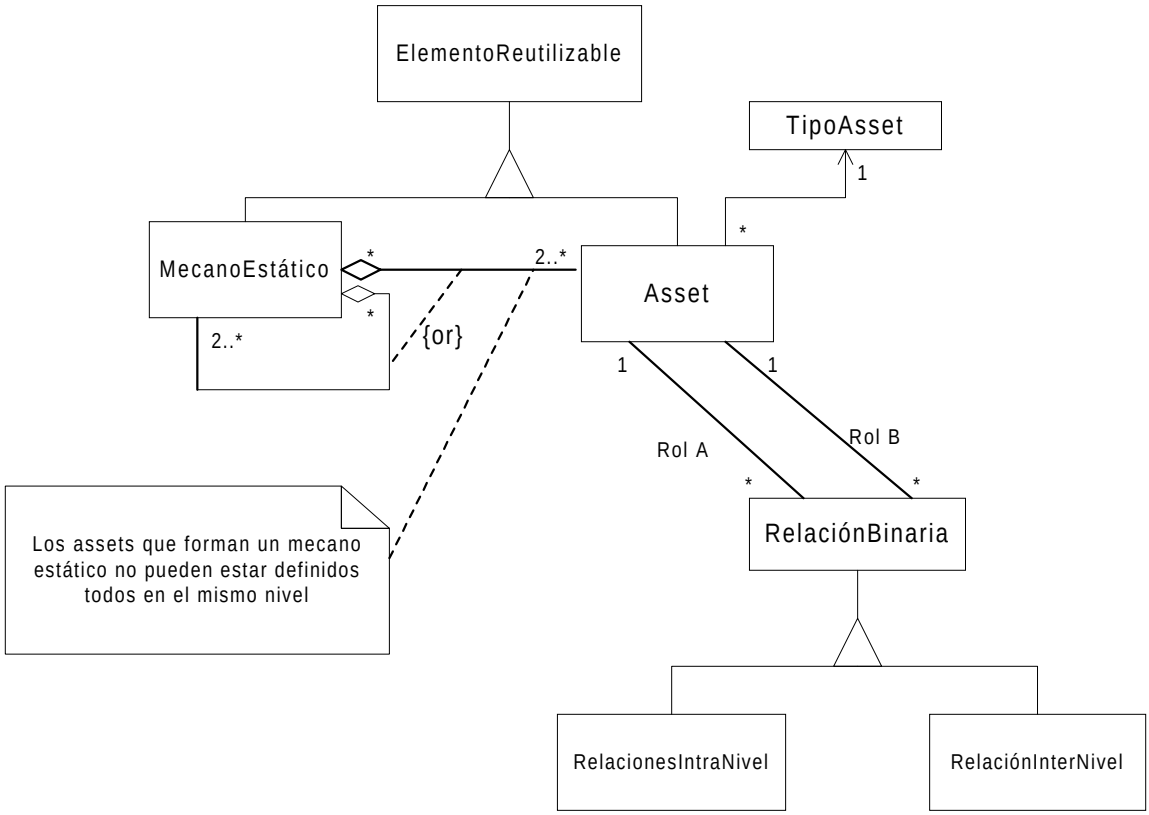

Figura 9. Modelo de mecano estático.

6. Agradecimientos

Este documento ha sido generado en el seno del grupo GIRO (*Grupo de Investigación en Reutilización y Orientación al Objeto*) compuesto por miembros del Departamento de Informática de la Universidad de Valladolid y del Área de Lenguajes y Sistemas Informáticos de la Universidad de Burgos. Desde aquí queremos agradecer la inestimable colaboración y las correcciones sugeridas por el resto de los miembros de este grupo.

Este trabajo ha sido parcialmente financiado por el proyecto CICYT TIC97-0593-C05-05.

7. Referencias

[Abrial, 1974] Abrial, J. R. "*Data Semantics in Database Management Systems*". J.W. Klimbie and K. L. Koffeman, eds., North Holland, 1974.

[Ader et al., 1990] Ader, M., Nierstrasz, O., McMahon, S., Muller, G. and Pröfrock, A-K. "*The ITHACA Technology: A Landscape for Object-Oriented Appliction Development*". In the proceedings of ESPRIT'90 Conference. Kluwer Academic Publisher. November, 1990.

[Bellinzona et al., 1993] Bellinzona, R., Fugini, M. G. and de Mey, V. "*Reuse of Specifications and Designs in a Development Information System*". Proceedings of the IFIP WG 81 Conference on Information System Development Process Como, Italy. September 1993.

[Biggerstaff, 1992] Biggerstaff, Ted J. "*An Assessment and Analysis of Software Reuse*". Advances in Computers. Academic Press, Inc. Edited by Marshall C. Yovits. Vol. 34: 1-57. 1992.

[Booch, 1994] Booch, Grady. "*Object-Oriented Analysis and Design with Applications*". 2nd Ed.Benjamin Cummnings, 1994.

[Bracchi et al., 1976] Bracchi, G., Paolini, P. and Pelagatti, G. "*Binary Logical Associations in Data Modelling in Modelling in Data Base Management Systems*". G.M. Nijssen, ed., North-Holland Publishing. 1976.

[Browne, 1996] Browne, Shirley. "*The National HPCC Software Exchange Repository Planning Guide*". University of Tennessee. August 28, 1996.

[Cattell and Barry, 1997] Cattell, R.G.G. and Barry, Douglas. "*The Object Database Standard: ODMG 2.0*". Morgan Kaufmann. 1997.

[Constantopoulos et al., 1992] Constantopoulos, Panos, Jarke, Matthias, Mylopoulos, Jonh and Vassiliou, Yannis. "*The Software Information Base: A Server for Reuse*". ITHACA.FORTH.92.E2#1. 1992.

[Cybulski, 1995] Cybulski, Jacob L. "*The Art of Reusing Software Requirements and Specifications: A Survey of Reuse Methods, Techniques and Tools*". http://leopard.lat.oz.au/~jacob/REUSE/c2reuse.html. 1995.

[Cybulski et al., 1997a] Cybulski, Jacob L., Neal, Ralpg D., Kram, Anthony and Allen, Jeffrey. "*Report on the Reuse of Early Life-Cycle Artefacts*". WISR8. Ohio. 1997.

[Denker, 1995] Denker, Grit. "*Transactions in Object-Oriented Specifications*". In E. Astesiano, G. Reggio, and A. Tarlecki, editors, Recent Trends in Data Types Specification, Proc. 10th Workshop on Specification of Abstract Data Types joint with the 5th COMPASS Workshop, S.Margherita, Italy, May/June 1994, Selected papers. Springer, Berlin, LNCS 906, 1995.

[Denker, 1996] Denker, Grit. "*Reification - Changing Viewpoint but Preserving Truth*". In Haveraan, M., and Owe, O., and Dahl, O.-J., editors, Recent Trends in Data Types Specification, Proc. 11th Workshop on Specification of Abstract Data Types joint with the 8th General COMPASS Meeting. Oslo, Norway, September 1995. Selected papers, pages 182-199. Springer, LNCS 1130, 1996.

[Denker and Ehrich, 1995] Denker, Grit and Ehrich, Hans-Dieter. "*Action Reification In Object Oriented Specification*". In R. J. Wieringa and R. B. Feenstra, editors, Information Systems - Correctness and Reusability, Selected Papers from the IS-CORE Workshop, pp. 103-118. World Scientific, 1995.

[DOD, 1992] DOD. "*DoD Software Reuse Initiative Vision and Strategy*". DOD, 1992.

[Edwards et al., 1997] Edwards, Stephen H., Gibson, David S., Weide, Bruce W. and Zhupanov, Sergey. "*Software Component Relationships*". WISR8. Ohio. 1997.

[Eichmann, 1995] Eichmann, David. "*The Repository Based Software Engineering Program*". In the Proceedings of the Fifth Systems Reengineering Technology Workshop, Monterrey, CA. February 7-9, 1995.

[Eichmann et al., 1995] Eichmann, David, Price, Margaretha, Terry, Robert H. and Welton, Louann L. "*ELSA and MORE: A Library and an Environment for the Web*". http://rbse.mountain.net/MOREplus/ELSAandMORE. 1995.

[Firesmith et al. 1996] Firesmith, Donald, Henderson-Sellers, Brian, Graham, Ian and Page-Jones, Meilir. "*OPEN Modeling Language (OML) Reference Manual*". Version 1.0, OPEN Consortium, 8 December 1996.

[Freeman, 1987a] Freeman, P. 1987a. "*Reusable Software Engineering: Concepts and Research Directions*". In Tutorial: Software Reusability, P. Freeman editor: 10-23.

[García et al., 1997a] García, Francisco José, Marqués, José Manuel y Maudes, Jesús Manuel. "*Mecano: Una Propuesta de Componente Software Reutilizable*". In the proceedings of the II Jornadas de Ingeniería del Software (Donostia-San Sebastián, Spain, 3-5 septiembre de 1997): 232-244. 1997.

[García et al., 1997b] García, Francisco José, Marqués, José Manuel and Maudes, Jesús Manuel. "*Análisis y Diseño Orientado al Objeto para Reutilización*". Technical Report (TR-GIRO-01-97V2.1), Valladolid University (Spain). Octubre 1997.

[García et al., 1998] García Peñalvo, Francisco José, Marqués Corral, José Manuel, Laguna, Miguel Ángel y Maudes Raedo, Jesús Manuel. "*Estructuras Complejas de Reutilización: Definición de Mecano Estático*". En las actas de las II Jornadas de Trabajo MENHIR. Editor José A. Carsí (Valencia, 19-20 de Febrero de 1998): 135-141. 1998.

[Girow, 1996] Girow, Andrew. "*Objects and Binary Relations*". Object Currents. Vol.1, Issue 6, SIGS Publications, June 1996.

[Griss, 1993] Griss, Martin L. "*Software Reuse: From Library to Factory*". IBM Systems Journal 32(4), 1-23, November, 1993.

[Griss and Kessler, 1996] Griss, Martin L. and Kessler Robert R. "*Building Object-Oriented Instrument Kits*". Object Magazine, April 1996:71-81. 1996.

[Griss and Wentzel, 1995a] Griss, Martin L. and Wentzel, Kevin D. "*Hybrid Reuse with Domain-Specific Kits*". In WISR'93:6th Annual Workshop on Software Summary and Working Group Reports. Edited by Jeff Paulin and Will Tracz. 1995.

[Griss and Wentzel, 1995b] Griss, Martin L. and Wentzel, Kevin D. "*Hybrid Domain-Specific Kits*". J. Systems Software, 30:213-230. 1995.

[Haase, 1996] Haase, Ken. "*Invention and Exploration in Discovery*". PhD Thesis. MIT Media Laboratory. 1996.

[Huhn et al., 1995] Huhn, M., Wehrheim, H., and Denker, G. "*Action Refinement - An Application of Process Theory on Object-Oriented Specification*". Hildesheimer Informatik-Berichte 40/95, Universität Hildesheim, Institut für Informatik, Postfach 101363, D-31113 Hildesheim, November 1995.

[Huhn et al., 1996] Huhn, M., Wehrheim, H., and Denker, G. "Action Refinement in System Specification: Comparing a Process Algebraic and an Object-Oriented Approach.. In U. Herzog, H. Hermanns, editors,GI/ITG-Fachgespräch: ``Formale Beschreibungstechniken für verteilte Systeme", 20/21. Juni 1996, Universität Erlangen, Germany, number 29/9 in Arbeitsbericht des IMMD, pages 77-88, 1996.

[IEEE, 1995] IEEE. "*IEEE Standard for Information Technology – Software Reuse – Data Model for Reuse Library Interoperability Data Model (BIDM)*". IEEE Std 1420.1, 1995.

[Johannsson et al., 1996] Johannsson, Paul, Boman, Magnuns, Bubenko, Janis jr. and Wangler, Benkt. "*Conceptual Modelling*". Prentice-Hall. http://www.dsv.su.se/~vadim/cmnew/index.htm. 1996.

[Jones, 1984] Jones, T. Casper. "*Reusability in Programming*: *A Survey of the Stateof the Art*". IEEE Trans. Software Engineering, Vol. 10, Nº5 (September): 488-494. 1984.

[Karlsson, 1995] Karlsson, Even-André. "*Software Reuse. A Holistic Approach*". John Wiley & Sons Ltd. 1995.

[Katz, 1990] Katz, R. "*Toward a Unified Framework for Version Modelling in Engineering Databases*". ACM Computing Surveys. Vol. 22, Nº 4: 375-408. 1990.

[Krueger, 1992] Krueger, Charles W. "*Software Reuse*". ACM Computing Surveys. Vol. 24. Nº 2: 131-183. 1992.

[Madany et al., 1991] Madany, Peter W., Islam, Nayeem, Kougiouris, Panos and **Campbell, Roy H.** "*Reification and Reflection in C++: An Operating Systems Perspective*". Technical Report, Department of Computer Science, University of Illinois at Urbana-Champaign, 1304 W. Springfield Avenue, Urbana, IL 61801, USA, 1991.

[Martin, 1996] Martin, Robert C. "*Granularity*". C++ Report. November-December 1996.

[Martínez y Maudes, 1997] Martínez Jiménez, Beatriz y Maudes Raedo, Margarita. "*Desarrollo de Componentes Software Reutilizables para el Dominio del Tratamiento Digital de Imágenes*". Proyecto Fin de Carrera de la Ingeniería Técnica en Informática de Sistemas de la Universidad de Valladolid. Departamento de Informática (ATC, CCIA, y LSI) Escuela Universitaria Politécnica de Valladolid. Dirigido por José Manuel Marqués Corral y Francisco José García Peñalvo. Septiembre, 1997.

[McClure, 1997] McClure, Carma. "*Software Reuse Techniques: Adding Reuse to the System Development Process*". PrenticeHall. 1997.

[Metamodel, 1997] "*Metamodelling Glossary*". http://www.metamodel.com/glossary.html. 1997.

[Meyer, 1994] Meyer, Bertrand. "*Reusable Software. The Base Object-Oriented Component Libraries*". Prentice-Hall, 1994.

[Meyer, 1997] Meyer, Bertrand. "*Object Oriented Software Construction*". 2nd Edition. Prentice Hall, 1997.

[Mili et al., 1995] Mili, Hafedh, Mili, Fatma and Mili, Ali. "*Reusing Software: Issues and Research Directions*". IEEE Transactions on Software Engineering. Vol. 21. Nº 6 (June): 528-562. 1995.

[NIST, 1994] National Institute of Standards and Technology (NIST). "*Glossary of Software Reuse Terms*". NIST, http://sw-eng.falls-church.va.us/ReuseIC/pubs/reference/terminology.htm, December 1994.

[Parnas et al., 1989] Parnas, D. L., Clements, P. C., Weiss, D. M. 1989. "*Enahancing Reusability with Information Hiding*". In Software Reusability. Vol 1 Concepts and Models, Ted J. Biggerstaff and Alan J. Perlis eds., ACM Press. Frontier Series.

[Peuker, 1997] Peuker, Thomas. "*An Object-Oriented Architecture for Real-Time Transmission of Multimedia Data Streams*". Institut für Mathematische Maschinen und Datenvararbeitung (Informatik) IV. Universität Erlangen-Nürnberg. http://wwwcip.informatik.uni-erlangen.de/~tspeuker/papers/book_html. 1997.

[Plaza, 1997] Plaza, Enric. "*Noos Language*". http://www.iiia.csic.es/~enric/noos/Overview. Draft, January 23, 1997.

[Rational et al., 1997a] Rational Software Corporation, Microsoft, Hewlett-Packard, Oracle, Sterling Software, MCI Systemhouse, Unisys, ICON Computing IntelliCorp, i-

Logix, IBM, ObjecTime. Platinum Technology, Ptech, Taskon, Reich Technologies and Softeam. "*UML Summary*. *Version 1.1*". UML 1.1 Referece Set 1.1. 1 September 1997**.**

[**Rational et al., 1997b] Rational Software Corporation, Microsoft, Hewlett-Packard, Oracle, Sterling Software, MCI Systemhouse, Unisys, ICON Computing IntelliCorp, i-Logix, IBM, ObjecTime. Platinum Technology, Ptech, Taskon, Reich Technologies and Softeam**. "*UML Semantics*. *Version 1.1*". UML 1.1 Referece Set 1.1. 1 September 1997.

[Rational et al., 1997c] Rational Software Corporation, Microsoft, Hewlett-Packard, Oracle, Sterling Software, MCI Systemhouse, Unisys, ICON Computing IntelliCorp, i-Logix, IBM, ObjecTime. Platinum Technology, Ptech, Taskon, Reich Technologies and Softeam. "*UML Notation Guide*. *Version 1.1*". UML 1.1 Referece Set 1.1. 1 September 1997.

[Rivas et al., 1997] Rivas, Erick, DeSilva, Dilhar, McDaniel, Terrie and Atkinson, Colin. "*Object Analysis and Design Facility*". Response to OMG/OA&D RFP-1. Version 1.0. Platinum Technology, Inc. January, 1997.

[NHSE, 1997a] NHSE. "*The Internal Data Format of RIB*". NHSE. October 24, 1997.

[NHSE, 1997b] NHSE. "*Repository in a Box (RIB) User's Guide*". Version 1.0. NHSE. 1997.

[Rumbaugh et al., 1991] Rumbaugh, James, Blaha, Michael, Premerlani, William, Eddy, Frederick and **Lorensen, William**. "*Object-Oriented Modeling and Design*". Prentice-Hall, 1991.

[SAIC, 1997] SAIC. "*ASSET FAQ*". SAIC. http://www.asset.com/WSRD/faq.html. 15 January 1997.

[Sema Group, 1996] Sema Group. "*Euroware User's Manual*". Sema Group. 1996.

[SER, 1996] SER Consortium. "*Solutions for Software Evolution and Reuse*". SER Esprit Project 9809. January, 1996.

[Trump, 1997] Trump, Daniel. "*Using the WWW and the Internet to Support Corporate Reuse*". WISR 8. Ohio. 1997

[Villa, 1997] Villa Esther. "*Estudio y Mejora de un Repositorio de Software*". Proyecto Fin de Carrera de la Ingeniería Informática de la Universidad de Valladolid. Departamento de Informática (ATC, CCIA, y LSI) Facultad de Ciencias de la Universidad de Valladolid. Dirigido por Pablo de la Fuente y José Manuel Marqués. Septiembre, 1997.

[Web, 1997] Principia Cybernetica Web. "*Web Dictionary of Cybernetics and Systems*". http://pespmc1.vub.ac.be/ASC. 1997.

[Webster, 1996] Merriam-Webster Inc. "*Webster's Third New International Dictionary*". Merriam-Webster, 1996.