Mecanos: Soporte de Diferentes Niveles de Abstracción en los elementos Software Reutilizables

Versión 1.2.1 Octubre de 1998

TR-GIRO-02-98V1.2.1

Francisco José García Peñalvo [1]
e-mail: fgarcia@gugu.usal.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 [3]
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 centrando todos los esfuerzos en la reutilización del código fuente y generalmente empleando técnicas intuitivas, lejanas a cualquier inicio de sistematización. 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 elemento 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 un elemento reutilizable ofrezca un soporte simultáneo de diferentes niveles de abstracción. Este elemento reutilizable 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, estructura compleja de reutilización, elemento reutilizable, asset, mecano, relación estructural, relación semántica.

---

[1] Departamento de Informática y Automática. Facultad de Ciencias, Universidad de Salamanca. Plaza de la Merced S/N, 37008 Salamanca (España).

[2] Departamento de Informática. Edificio de Tecnologías de la Información y las Telecomunicaciones. Universidad de Valladolid. Campus Miguel Delibes, 47011 Valladolid (España).

[3] 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).

Tabla de Contenidos

1. Antecedentes al presente trabajo

El presente informe técnico sustituye completamente al informe técnico **TR-GIRO-01-98V0.9** cuyo título es *Mecanos Reutilizables Estáticos*. Los contenidos del presente documento representan una importante evolución en el trabajo de investigación que se está acometiendo, dejando obsoletas algunas de las bases que se planteaban en el documento **TR-GIRO-01-98V0.9**, actualizándose además el estado del arte con las últimas referencias consultadas.

2. 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 cabe destacar los siguientes hechos:

- *La desconfianza en el trabajo realizado por otras personas*[1] [McClure, 1997].
- *Falta de soporte a la reutilización por parte de los métodos 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 incremento de la potencia y del alcance del proceso de reutilización del software 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.

Como consecuencia se puede deducir que la reutilización debe enfocarse hacia la reutilización de elementos software que aporten más información que el código fuente (*como ejemplo se puede afirmar que el nivel de implementación no es capaz de retener toda la información del diseño, produciéndose una pérdida importante cuando sólo se produce una reutilización del código fuente*). Esto lleva a plantarse la reutilización de los elementos software producidos en las primeras fases del ciclo de vida del desarrollo del software (*elementos relacionados con los requisitos y los diseños de alto nivel*) con toda la problemática que esto conlleva.

---

[1] Efecto numerosamente citado en la literatura sobre reutilización con el nombre de *síndrome de no inventado (desarrollado) aquí.*

Surge así un nuevo concepto para reflejar el alcance actual de la reutilización, ***el asset***[2]. 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**[3].

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***.

---

[2] Existe una cierta controversia con el término asset para hacer referencia a los elementos software reutilizables. En la literatura especializada en reutilización aparecen una serie de términos que hacen referencia al mismo concepto: componente software reutilizable, artefacto software reutilizable y asset, recomendándose por lo sobrecargado del término componente la utilización de asset y así evitar las ambigüedades propias de la sobrecarga de significados. Sin embargo, al intentar traducir la palabra asset al español aparece un problema, no existe un término adecuado para sustituir a la traducción literal (valor) que no es muy adecuada en el contexto de la Ingeniería del Software (más concretamente en el ámbito de la Reutilización del Software). Tampoco es adecuada la utilización del concepto de elemento software reutilizable por las características del trabajo llevado a cabo donde se quiere distinguir entre unos elementos reutilizables de granularidad fina (assets) y unos elementos de granularidad gruesa (mecanos) cuya definición es el objetivo del proyecto de investigación que actualmente está llevando a cabo el Grupo de Investigación GIRO (Grupo de Investigación en Reutilización y Orientación a Objetos).

Con objeto de poner fin a la polémica terminológica, al menos en el seno del grupo GIRO, se ha optado por utilizar el vocablo **asset** sin traducir, y con el significado de elemento software reutilizable de granularidad fina e independiente del nivel de abstracción en el que está clasificado.

[3] 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 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 no formal [García et al., 1997a], [García et al., 1998a], incidiendo muy especialmente en las relaciones entre los assets que forman los elementos software reutilizables de mayor granularidad [García et al., 1998b].

El resto del documento se organiza como sigue. En la sección 3 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 cuarta sección se presenta la definición del concepto de mecano como elemento software reutilizable multinivel de grano grueso. En esta sección se presentará una lista con los requisitos que debe cumplir esta estructura. La sección 5 está dedicada por completo a la descripción del modelo de mecano, presentando un modelo que representa su estructura. En la sección 6 se introduce el proceso de desarrollo con reutilización desde el punto de vista de los mecanos. Las conclusiones y el trabajo futuro cierran este documento en la sección 6.

3. 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[4] 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.

3.1 Desarrollo software

El objetivo de una estructura compleja de reutilización, al menos en un primer momento, se centra en dar soporte para su futura utilización en ámbitos diferentes al inicialmente considerado, a un esfuerzo invertido en el desarrollo software concreto. 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, componentes ActiveX*...), constituye un esfuerzo de desarrollo software perfectamente válido, y del cual se puede derivar un mecano como elemento reutilizable complejo.

[4] 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.

3.2 Asset

En el presente trabajo se está definiendo un elemento software reutilizable complejo, definido en varios niveles de abstracción. Pero, esta estructura reutilizable puede verse como un *elemento software reutilizable especial* de mayor granularidad[5], que acoge en su seno un conjunto de assets de menor granularidad relacionados entre sí, los cuales se encuentran clasificados en una serie de niveles de abstracción.

Según esto, se puede definir ***asset*** *como aquel elemento software, que con independencia de su nivel de abstracción, ha sido seleccionado y preparado 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 distintos al de su creación, o bien en el esfuerzo de su identificación, extracción y configuración de proyectos software ya terminados [Karlsson, 1995]*.

Los diferentes assets normalmente comparten una serie de características que influyen activamente en la potenciación de su reutilización [Cybulski, 1995], [Cybulski, 1996]:

- *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.
- *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.

[5] 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].

- *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 assets deben omitir los detalles de implementación propios de los lenguajes de programación. Esto es, los assets deben 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.

Pero antes de adentrarnos en el estado del arte de la estructura de los assets conviene pararse antes en aclarar qué se entiende en este trabajo por repositorio, pues de nuevo se está ante una palabra de múltiple aceptación en Ingeniería del Software, pero con múltiples acepciones de significado.

3.2.1 Repositorios y bibliotecas de software

El proceso de producción de un determinado sistema software mediante reutilización sólo tiene sentido si existe un enlace entre el desarrollo para reutilización, donde los assets son producidos, y el desarrollo con reutilización, donde éstos son utilizados. Esto lleva a la necesidad de contar con un almacén de assets que enlace los dos procesos.

Estos almacenes se conocen como repositorios de reutilización, constituyéndose en elementos centrales para el soporte operativo de la reutilización.

No obstante, el concepto de repositorio es un concepto amplio que va desde sencillos sistemas de almacenamiento hasta complejos entornos que incorporan, además de los sistemas de almacenamiento, conjuntos de herramientas de ayuda al proceso de reutilización.

Debe tenerse presente que un repositorio no es un fin en sí mismo, sino un soporte al proceso de reutilización, de forma que el esquema del repositorio ha de responder al modelo de elemento software reutilizable y al tipo de reutilización adoptado por la organización que lo pone en funcionamiento.

En [Bernstein and Dayal, 1994] se define repositorio como *una base de datos de información compartida sobre los elementos que se producen o se usan en un desarrollo software.*

Inicialmente el concepto de repositorio se corresponde con una simple base de datos para el almacenamiento de assets. Sin embargo, el concepto de repositorio evoluciona hacia entornos más sofisticados, con complejos métodos de almacenamiento, búsqueda, navegación, examen detallado de los assets almacenados y recuperación [Kara, 1997], [Viasoft, 1997], [Durnin, et al., 1996].

Esta evolución del concepto de repositorio casa con el concepto de biblioteca de reutilización propugnado por el **DoD** (*Department of Defense*) de **EEUU** y los estándares de la **OTAN** para reutilización. Así, una biblioteca de reutilización se puede definir como *una colección de elementos software reutilizables, junto a los procedimientos y funciones de soporte requeridas para ofrecer los assets a los usuarios* [NATO, 1992].

De esta forma en la literatura especializada se alternan los términos repositorio y biblioteca, aunque con idéntico significado. Para una mayor información sobre los repositorios se recomienda la consulta de [Marqués, 1998].

3.2.2 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].

A modo de resumen en el Cuadro 1 se recogen los elementos más destacados de **ITHACA**.

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 elemento 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 elemento 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].

- Es una referencia obligada para cualquier trabajo de reutilización en varios niveles de abstracción.
- No ofrece una definición precisa de lo que es la estructura de un elemento 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 elementos reutilizables.

Cuadro 1. Resumen de las características del elemento reutilizable de ITHACA.

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.

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 asset, soluciones a dichos problemas… de forma que se cuenta con una especie de histórico de la reutilización del mismo. A cada asset le corresponde su propia cualificación.

La ***información administrativa*** hace referencia a la información general del asset[6], 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 assets, pero cada asset sólo puede tener una información administrativa.

[6] 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.

La ***documentación*** es esencial en la reutilización de un elemento reutilizable. 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 asset es uno a varios.

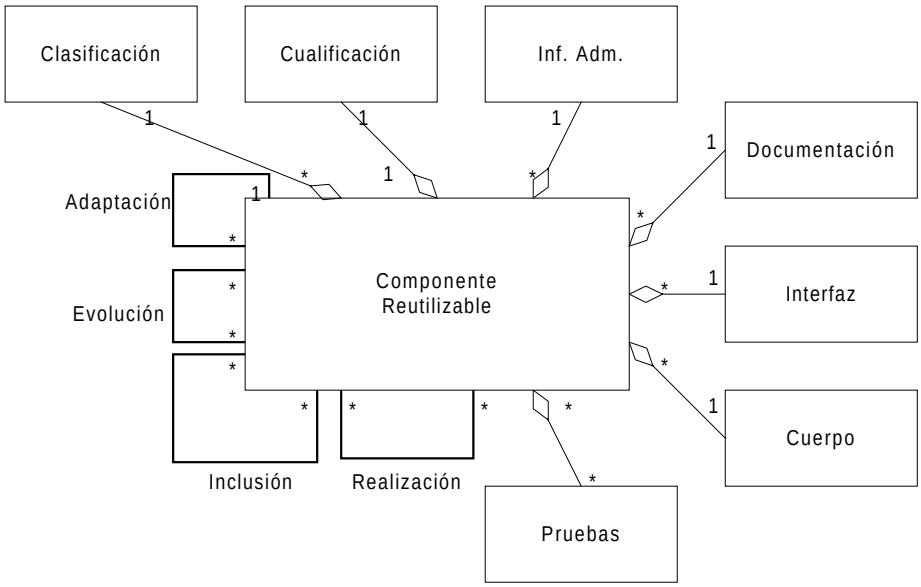

Figura 1. Modelo de elemento reutilizable de REBOOT

La ***interfaz*** del asset describe la forma de comunicarse con el asset, 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 asset 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 asset y las pruebas es varios a varios, ya que un asset puede tener varios juegos de prueba, y varios juegos de prueba se pueden aplicar a varios assets.

La ***relación de realización*** relaciona assets clasificados en diferentes niveles de abstracción. De esta forma, si se tiene que reutilizar una especificación, se podría acceder al diseño e implementación correspondientes, ahorrando el trabajo en las fases

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

Una vez que se ha realizado este repaso por el modelo de asset propuesto por REBOOT, y a modo de resumen, en el Cuadro 2 se recogen las principales conclusiones sobre esta iniciativa.

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.

- 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 [illegible] de cualquier connotación [illegible] assets.

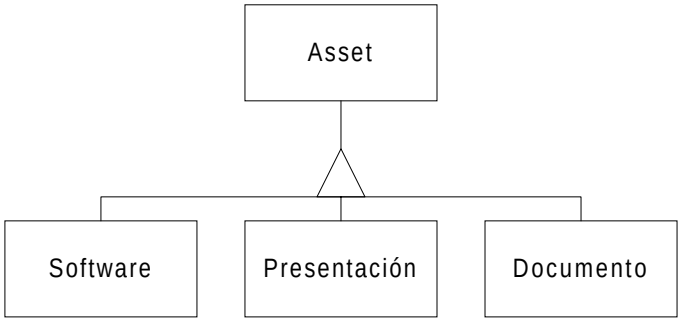

Figura 2. Jerarquía de assets en EUROWARE.

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.**

En el Cuadro 3 se presentan las conclusiones que se sacan del estudio realizado sobre el modelo de datos de **EUROWARE**.

- **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 introducir el concepto de un elemento reutilizable con soporte simultáneo de diferentes niveles de abstracción en el que los enlaces entre sus componentes atómicos careciesen de contenido semántico.

Cuadro 3. Resumen de las características del elemento reutilizable de EUROWARE.

El programa **STARS** (*Software Technology for Adaptable, Reliable Systems*) patrocinado por la agencia **DARPA** (*Defense Advanced Research Projects Agency*) ha sido uno de los esfuerzos más importantes de investigación en la reutilización del software llevado a cabo bajo la supervisión del **DoD**. Aunque **STARS** es una fuente de importantes referencias en reutilización, cabe destacar especialmente dos aspectos de este proyecto: **CFRP** (*Conceptual Framework for Reuse Processes*) y **ALOAF** (*Asset Library Open Architecture Framework*).

El **CFRP** define un contexto para entender, definir e integrar los procesos de ingeniería del software relacionados con la reutilización desde la doble perspectiva de la gestión y los aspectos técnicos. Con el **CFRP** se puede ofrecer una base para la adopción y mejora de las prácticas de reutilización dentro de una organización. Sin embargo, el estudio del **STARS CFRP** cae fuera de los objetivos del presente documento, siendo [Creps et al., 1992] una buena fuente para profundizar en él.

STARS ALOAF [Solderitsch, 1992], [Solderitsch et al., 1992] tiene como objetivo la definición de una forma de intercambio de assets entre diferentes repositorios[7], y la definición de una plataforma para repositorios sobre la cual construir herramientas

[7] Para ser más correctos con la nomenclatura utilizada por **STARS** en lugar de repositorio se debería utilizar el término **biblioteca de assets**, pero como ya se indicó al comienzo de este documento, en el ámbito del presente trabajo los términos repositorio y biblioteca de assets se consideran sinónimos.

portables de reutilización. Para ofrecer una base para establecer una comunicación entre los diversos repositorios orientados a la reutilización de assets, **ALOAF** incluye el **STARS ALF** (*Asset Library Framework*) **RM** (*Reference Model*) para repositorios.

El modelo de referencia ofrece una visión general de los repositorios y de los elementos que los componen. Va a establecer la terminología básica para repositorios, así como los principales principios en los que se basan los repositorios. Las características básicas del Modelo de Referencia de **STARS** se muestran en la Figura 3.

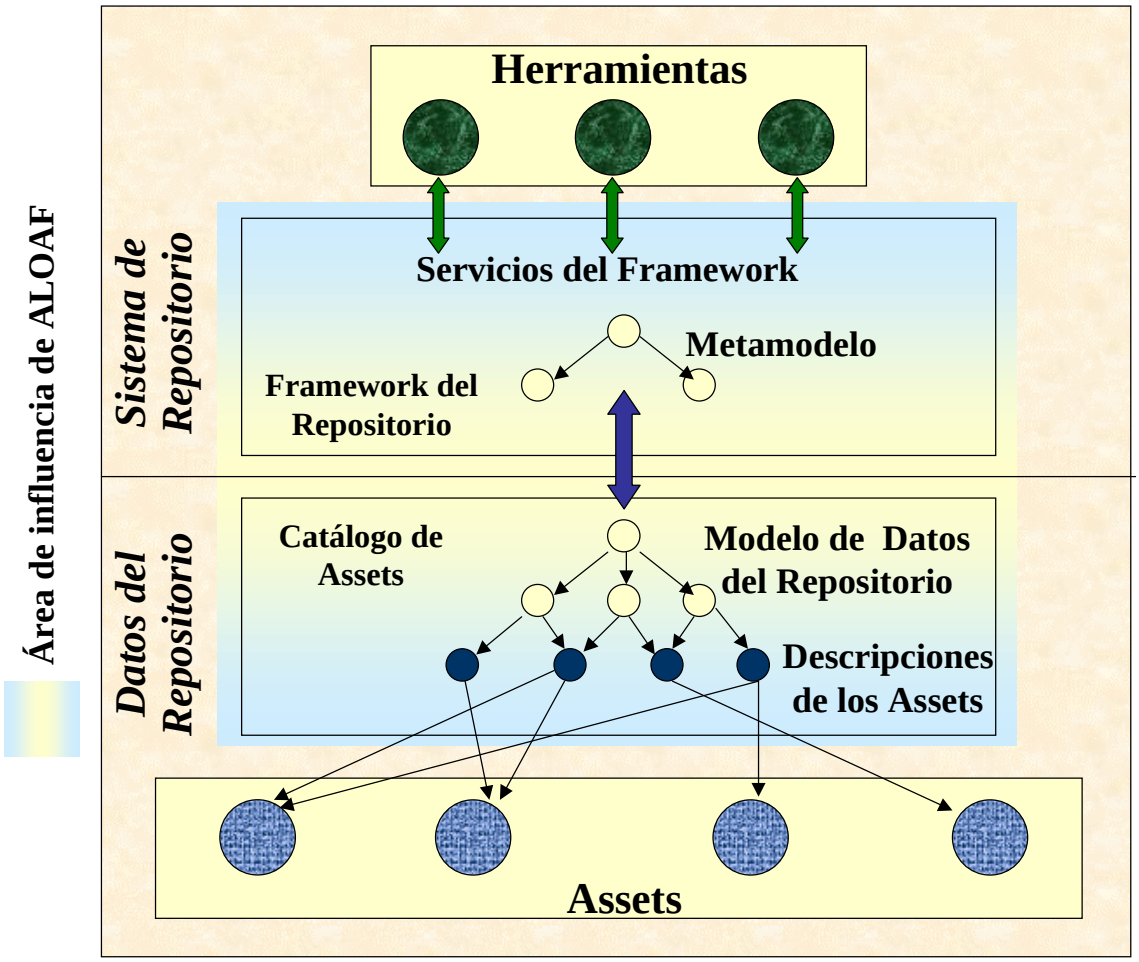

Figura 3. Repositorio según el Modelo de Referencia de STARS.

Un repositorio se correspondería con la caja exterior de la Figura 3. Un repositorio se compone de una parte de datos y de una parte de sistema. La parte de datos consiste en los propios assets, además de en una información descriptiva y de organización sobre los assets. Esta información es denominada **Catálogo de Assets** en la Figura 3. Este catálogo de assets contiene la información descriptiva (*Descripciones de los assets*) asociada con los assets y la organización estructural de las descripciones (*Modelos de datos del repositorio*).

La parte de sistema del repositorio ofrece un conjunto de mecanismos que operan sobre los datos del repositorio. El sistema del repositorio está formado por el framework del repositorio y los servicios que ofrecen las herramientas del repositorio. Los servicios se necesitan para crear y manejar el catálogo de assets de acuerdo a la técnica de estructuración de datos definida en el metamodelo.

El área en la que se centra **ALOAF** incluye el *Framework* y el *Catálogo* del repositorio, no incluyendo los assets ni las herramientas que usan los servicios del repositorio. El framework del repositorio sirve como guía de implementación de los repositorios y de las herramientas que se necesitan. El metamodelo define y limita las formas de construcción de los modelos de datos; describe como los modelos de datos se construyen utilizando bloques de construcción básicos. Por su parte el catálogo determina como se describen los assets y el formato de los componentes de una descripción de un asset.

Como se indicó al comenzar a hablar de **ALOAF**, uno de los principales objetivos perseguidos era facilitar el intercambio de assets entre repositorios, para lo cual **ALOAF** cuenta con una especificación de intercambio de assets y de descripciones de assets. Como primera solución para el intercambio de assets se propone un modelo común de datos, el **CDM** (*Common Data Model*). Este modelo de datos describe un modelo básico de datos que permite a los repositorios intercambiar los assets que cumplen un subconjunto común de descripciones. El **CDM** no es más que un conjunto de clases, con una clase en la base de la jerarquía denominada **Objeto** que contiene un par de atributos básicos (*identificador y nombre*) que son heredados por el resto de las clases del modelo. El **CDM** se muestra en la Figura 4.

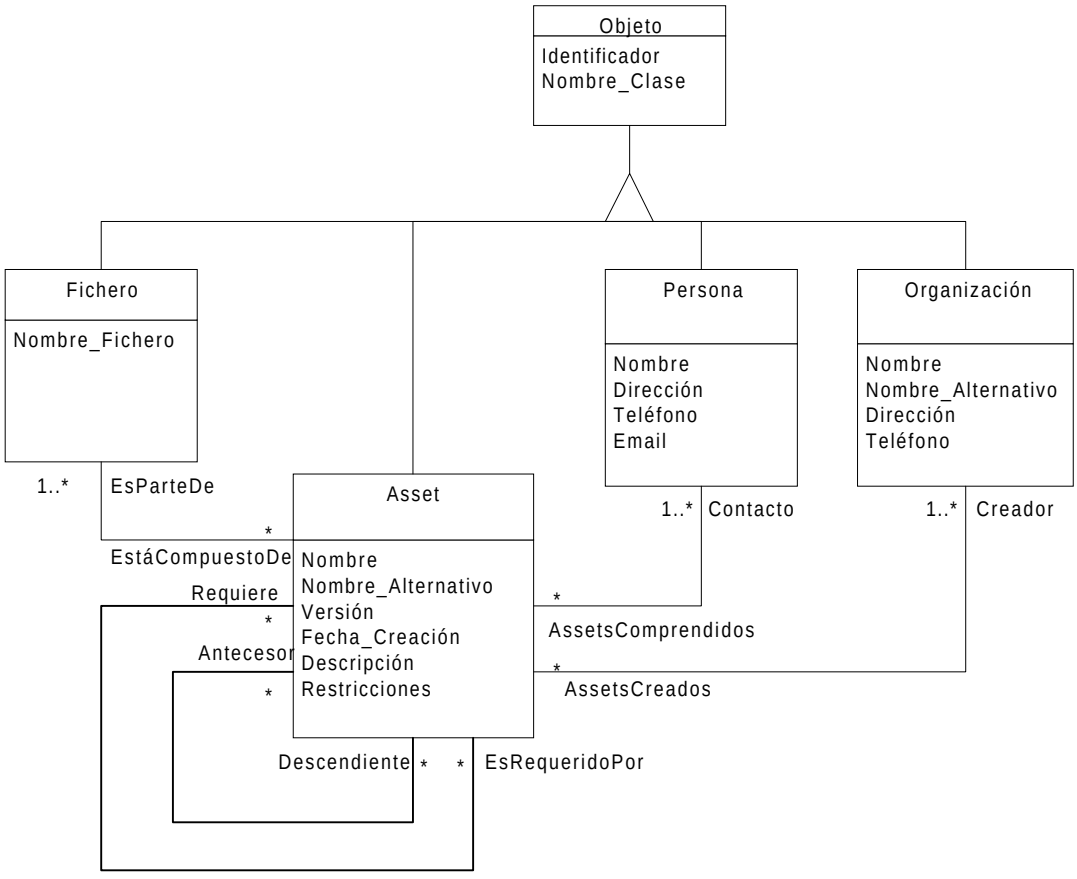

Figura 4. El modelo CDM de STARS.

El centro del **CDM** es la clase **Asset**, la cual contiene un conjunto de atributos que describen las propiedades generales de los assets. Las clases **Organización** y **Persona** ofrecen información sobre las organizaciones y las personas responsables de los assets. La clase **Fichero** existe para representar la información sobre los ficheros que

constituyen los contenidos de un asset. Al utilizar el **CDM**, los assets se representan como objetos instancia de las clases del modelo.

Según el **CDM** un asset típico puede estar representado por una sola instancia de la clase **Asset**, por una (o posiblemente más) instancias de la clase **Organización**, por una (o posiblemente más) instancias de la clase **Persona**, y por uno o más instancias de la clase **Fichero**. Todas estas instancias relacionadas a través de las relaciones del modelo.

Para concluir con el repaso que del programa de reutilización **STARS** se ha realizado, en el Cuadro 4 se han recogido un resumen de las características más destacadas de este proyecto (*más en concreto de **ALOAF***) por su posible repercusión en el trabajo en curso que se está presentando en este documento.

- El programa STARS es una referencia muy importante para los proyectos de investigación en reutilización del software, destacando (*además de por su influencia en otros programas de reutilización de él derivados*) por establecimiento de un *framework conceptual para los procesos de reutilización* **CFRP** y de un *framework para la arquitectura de repositorios* **ALOAF**, especialmente interesante para el presente trabajo debido al modelo de referencia para repositorios que incluye.
- **ALOAF** establece un modelo de referencia para repositorios, con el que se quiere lograr un intercambio de assets entre repositorios para lo cual dentro de este modelo de referencia se define el modelo común de datos, **CDM**.
- El **CDM** de **ALOAF** tiene una gran similitud con el modelo **BIDM** propuesto por el **RIG** (*Reuse Library Interoperability Group*). De hecho, el modelo propuesto por el equipo de **STARS ALOAF** incorpora resultados del **RIG** como consecuencia de una mutua colaboración[8]**.**
- En lo referente a las relaciones entre assets, el **CDM** no es un modelo especialmente rico, pero aporta dos ideas importantes: por un lado hace referencia al soporte de versiones de assets mediante la relación **ancestro/descendiente**, y por otra parte incorpora una relación que sin recibir ningún nombre específico hace hincapié en el carácter del enlace frente al proceso de reutilización del asset, al indicar que los asset relacionados con el que va a ser recuperado son necesarios para la reutilización efectiva del primero.

Cuadro 4. Resumen de las características de STARS.

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

[8] El equipo de **ALOAF** va estuvo en contacto con el **RIG** en lo referente a la consecución de un estándar de elemento reutilizable. Hasta la primavera de 1992, las actividades del equipo de **ALOAF** y del **RIG** fueron bastante complementarias, con algún grado de solapamiento. De hecho, las partes de intercambio de assets y la definición de las capacidades de un repositorio que soporte interoperatividad fueron dirigidas por el equipo de **ALOAF**, de forma que algunos resultados del **RIG** fueron incorporados a **ALOAF** cuando estuvieron listos.

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 5. Además, en [NHSE, 1997b] se incluye un glosario explicando todos los términos de **BIDM** y **ACF**.

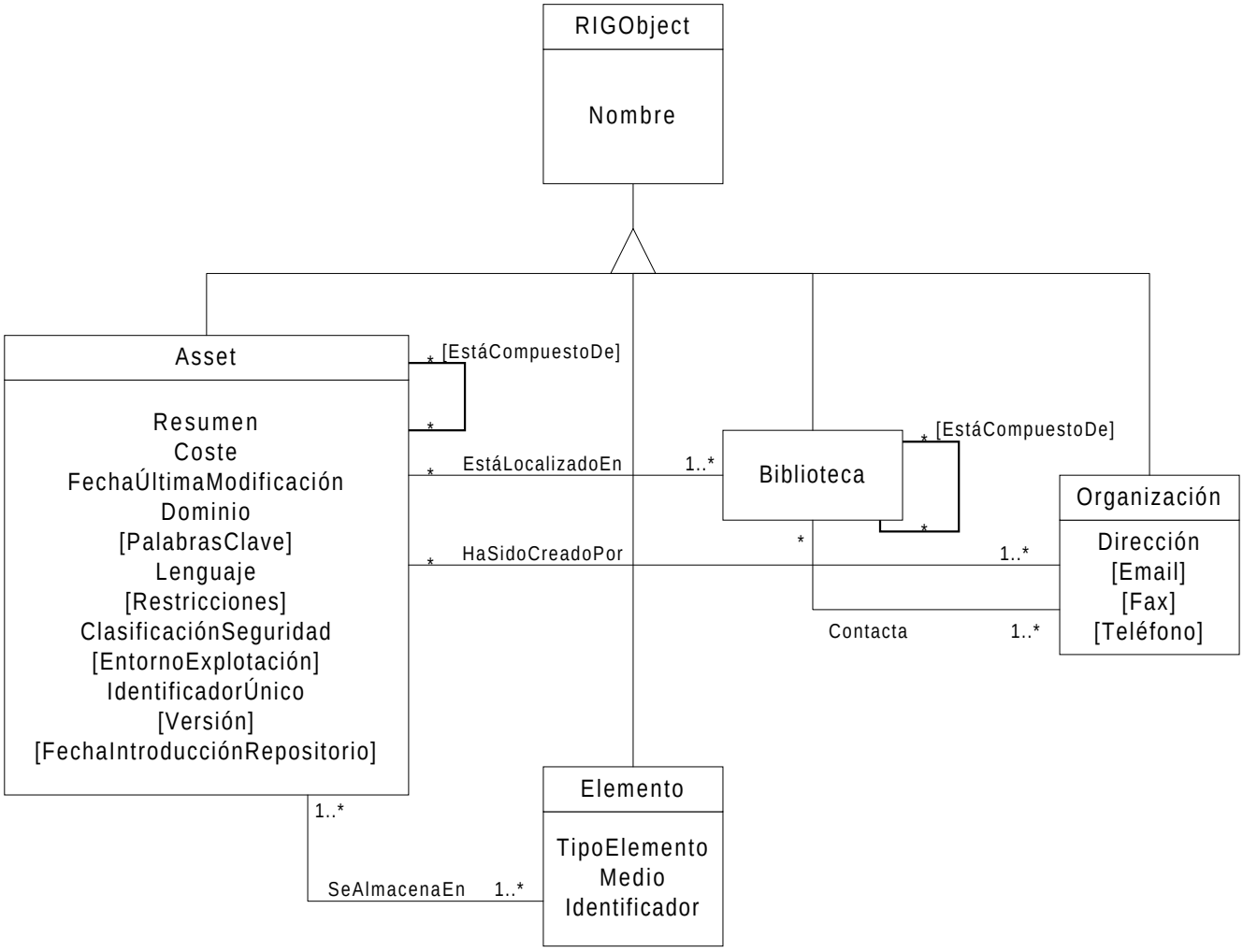

Figura 5. Basic Interoperatibility Data Model

Del estudio de **RIB** se pueden derivar un conjunto de importantes conclusiones sobre el modelo de asset. Estas conclusiones se recogen en el Cuadro 5.

- **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.

Cuadro 5. Resumen de las características del elemento reutilizable de RIB.

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 6.

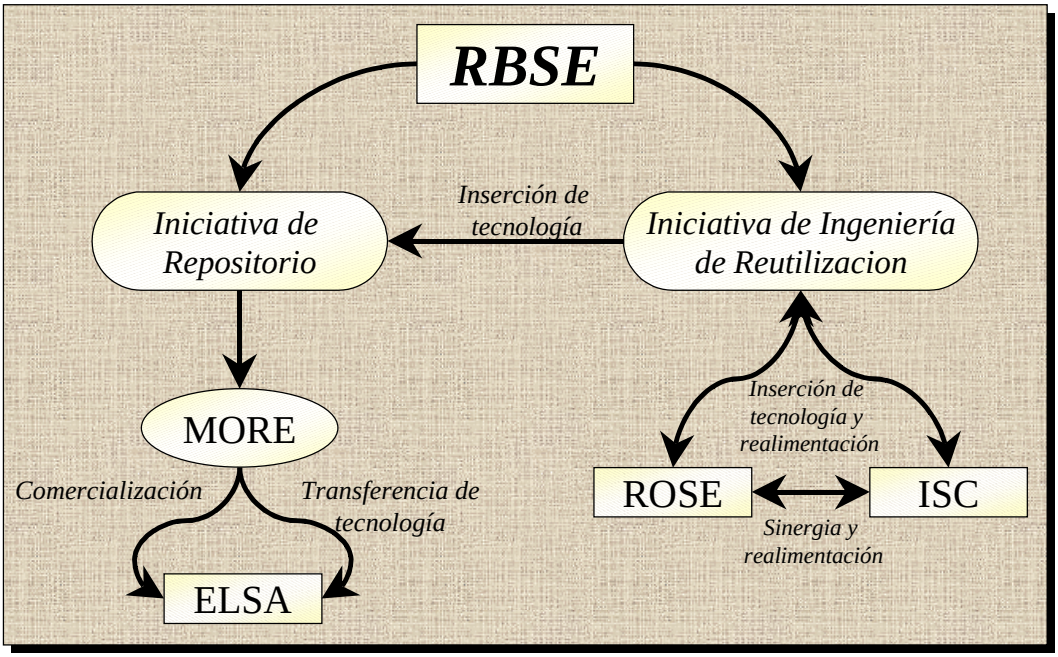

Figura 6. Arquitectura de RBSE.

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].

En el Cuadro 6 se recogen las características de **RBSE**, o más concretamente de **MORE/ELSA**, como resumen de este repositorio.

- **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**

repositorio que almacena componentes **COM**, y que se comercializa en la actualidad como un componente de Microsoft Visual BASIC 5.0.

El Repositorio Microsoft se compone de dos elementos principales: un conjunto de interfaces de componentes ActiveX y un motor de repositorio que sustenta el mecanismo de almacenamiento para los modelos de información que el usuario construye utilizando las interfaces.

La funcionalidad del motor del Repositorio Microsoft se soporta mediante un conjunto de objetos **COM** y Automation. El **COM** (*Component Object Model*) es el modelo de componente que sirve como base a la arquitectura de objetos propuesta por Microsoft [Rogerson, 1997], [López, 1997]. El **COM** es un estándar binario que describe las llamadas entre componentes, de forma que los componentes pueden estar escritos en un lenguaje y ser llamados desde un programa (*u otros componentes*) realizados en un lenguaje diferente.

Los objetos **COM** y Automation del Repositorio Microsoft son representaciones en memoria de la información almacenada en la base de datos del repositorio. Un objeto se puede considerar como un objeto del repositorio si soporta un cierto conjunto de interfaces específicas del repositorio.

El Repositorio Microsoft soporta cuatro tipos principales de objetos:

- **Sesión del repositorio:** Representa a la base de datos del repositorio propiamente dicha.
- **Objeto del repositorio:** Representa el estado persistente de un objeto en un repositorio. El estado consiste en las propiedades y colección del objeto.
- **Objeto colección:** Representa un conjunto de objetos relacionados. Una colección de relaciones es accedida y es actualizada utilizando los métodos estándar de la colección.
- **Objeto relación:** Representa una conexión entre dos objetos del repositorio. La relación puede tener propiedades. La relación y sus propiedades se almacenan en la base de datos del repositorio.

Las definiciones de tipos en el repositorio son objetos del repositorio que tienen ciertas propiedades y relaciones que son interpretadas por el motor del repositorio. Por ejemplo, una definición de una clase es un objeto que tiene una propiedad que representa su único identificador y una relación a las interfaces que implementa.

El motor del Repositorio Microsoft almacena sus datos en una base de datos relacional. Esta base de datos contiene las propiedades y relaciones de los objetos almacenados en el repositorio.

En el Cuadro 7 se recogen algunas de las principales características del Repositorio Microsoft.

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].

- El Repositorio Microsoft 1.0 es una implementación propietaria de un repositorio de componentes **COM** (*ActiveX*) accesibles desde Microsoft Visual BASIC 5.0.
- Aunque no aporta nada especial en cuanto a los modelos de elementos software reutilizables (*al utilizar como elementos reutilizables componentes es su estándar binario **COM***), si resulta interesante su estudio por las ideas que puede aportar a la hora de abordar la construcción de un repositorio.
- Es de destacar la forma en que se tratan las relaciones entre los objetos del repositorio, ya que son objetos a su vez que se almacenan en la base de datos.
- Resulta importante recalcar el hecho de que siendo el Repositorio Microsoft un repositorio orientado a objeto (*u orientado a componentes*), el motor del mismo se encuentra implementado sobre un motor de bases de datos relacional no orientado a objetos.
- Los objetivos perseguidos por este repositorio son principalmente dos. En primer lugar compatibilidad total con **COM** y Automation. Y en segundo lugar, asegurar la extensibilidad para clientes y vendedores software que necesitan construir el repositorio añadiendo funcionalidad a los objetos almacenados en el repositorio.

Cuadro 7. Resumen de las características del Repositorio Microsoft.

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** entre otros[10]. 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**[11] (*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.

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

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

- **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**[12].
- **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.
- **Función:** Qué hace el asset.
- **Objeto:** Objeto pasivo de la acción del asset.
- **Palabras clave:** Palabras clave, frases que describen aspectos importantes del asset.

En el Cuadro 8 se presenta un resumen sobre las características de **WSRD** de **SAIC/ASSET**.

• 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.

Cuadro 8. Resumen de las características de WSRD de SAIC/ASSET.

[12] 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**.

3.3 Nivel de abstracción

En el momento en que el ámbito de la reutilización se amplía para dar cobertura a elementos software 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 elementos reutilizables: *datos reutilizables, arquitecturas reutilizables, diseños reutilizables* y *programas reutilizables.*

En [Edwards et al., 1997] se propone una clasificación de los elementos relacionados con el proceso de la Ingeniería del Software en dos dimensiones ortogonales. La primera dimensión estaría formada por elementos abstractos (*especificaciones*) y por elementos concretos (*implementaciones*). La segunda dimensión está compuesta por elementos plantillas y por elementos 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*[13] [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.*

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

3.4 Relaciones entre assets

Un elemento reutilizable de granularidad gruesa con soporte de diferentes niveles de abstracción simultáneamente, puede verse como una malla de elementos de menor grano clasificados en diferentes niveles de abstracción. Para formar esta malla se tienen que establecer los enlaces entre los elementos de grano fino, enlaces que vienen caracterizados por las relaciones que se puedan establecer entre los tipos de assets que constituyen la malla.

Debido al soporte simultáneo de diferentes niveles de abstracción dentro del elemento reutilizable de grano grueso aparecen dos familias de relaciones entre los assets constituyentes: aquellas relaciones que se dan entre assets clasificados en el mismo nivel de abstracción – que serán las responsables de formar estructuras de reutilización de grano grueso con soporte de un solo nivel de abstracción – y aquellas relaciones que se dan entre assets clasificados en diferentes niveles de abstracción – que se constituyen en las bases estructurales para formar las estructuras de reutilización de grano grueso con soporte simultáneo de distintos niveles de abstracción, objetivo de este trabajo -.

Así pues, se va a realizar un repaso por la literatura para determinar la influencia de las relaciones semánticas entre assets en otros importantes trabajos sobre reutilización del software, pero haciendo una distinción entre las relaciones entre elementos clasificados en un mismo nivel de abstracción o en diferentes niveles de abstracción.

3.4.1 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 elementos 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 ***3.2.2 Estado del arte de la estructura de un asset***, se puede apreciar que las relaciones entre assets aparecen 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 semánticas estructurales que existirán entre assets clasificados en un mismo nivel de abstracción.

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.

El modelo **CDM** propuesto por **STARS ALOAF**, aunque no es un modelo especialmente rico en el tema de relaciones entre assets ni hace referencia explícita al nivel de abstracción de los assets, presenta dos ideas especialmente interesantes. La primera hace mención al versionado de assets y la segunda al carácter de obligatoriedad frente al proceso de reutilización de la relación **Requiere**, por la cual todos los assets relacionados con otro mediante esta relación son necesarios para la reutilización efectiva del primero.

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.

En el Repositorio Microsoft las relaciones entre los componentes soportados por el repositorio aparecen representadas como un objeto más del repositorio. En el Repositorio Microsoft las relaciones entre componentes son relaciones bidireccionales, que puede ser navegadas desde cualquiera de los dos objetos que enlazan. Como cualquier otro objeto del repositorio puede tener propiedades, aunque no puede tener ni métodos ni relaciones. Cada relación es una instancia de una *clase relación*. La definición de una clase relación conecta dos definiciones de colecciones llamadas origen y destino. Una instancia de una relación puede ser navegada en cualquier dirección, aunque algunas relaciones tengan connotaciones sensitivas a la polaridad de la relación indicada por el origen y el destino de la misma.

Además, en el Repositorio Microsoft una *clase relación* tiene tres mecanismos para dar soportes a las connotaciones semánticas de las relaciones: ***el nombre*** - toda relación puede tener un nombre que identifica al objeto destino con relación a su origen -***, la secuencia*** – dentro de una colección los objetos destino pueden tener una secuencia dentro del contexto de un objeto fuente particular - y ***la propagación de eliminación*** – los métodos de borrado pueden propagarse a objetos más allá de los que van a ser eliminados -.

De los modelos estudiados **ITHACA** y **REBOOT** son los modelos de referencia más importantes, aunque como implementación real el Repositorio Microsoft constituye un punto de referencia muy interesante. Sin embargo, el resto de los modelos estudiados tienen en las relaciones entre assets su asignatura pendiente. En general se pueden extrapolar una serie de conclusiones que se recogen en el Cuadro 9.

- En la mayoría de los repositorios el concepto de relación semántica aparece (*siendo inexistente en algún caso*) con la única connotación de enlace entre assets.
- La semántica de las relaciones suele recaer en el nombre de las mismas, utilizando normalmente nombres muy genéricos, no llevando restricciones de ningún tipo asociadas.
- La connotación semántica más difundida es la de composición, buscando representar distintos grados de granularidad en los assets; 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.
- **REBOOT** constituye un marco teórico de referencia para los modelos de assets con un conjunto de relaciones bastante completo, contemplando la relación de versionado a parte de las relaciones más típicas.
- El carácter de obligatoriedad frente al proceso de reutilización introducido por **ALOAF** es un aspecto muy interesante que se puede aprovechar en el tratamiento automático del proceso de reutilización.
- El Repositorio Microsoft aporta interesantes ideas sobre la forma de implementar las relaciones en un repositorio, así como en las características de éstas.

Cuadro 9. Conclusiones sobre el estudio de las relaciones entre assets en los modelos de assets.

Dado que los modelos estudiados son a todas luces insuficientes en el tema de las relaciones semánticas, se va a proceder a realizar un estudio de alguno de los diferentes modelos objeto[14] presentes en los principales métodos de desarrollo orientados a objeto.

Como referencias representativas se han seleccionado **OMT** [Rumbaugh et al., 1991], Método Booch [Booch, 1994], **UML 1.1** [Rational et al, 1997a], [Rational et al., 1997b], [Rational et al.,1997c] 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****.*

[14] 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.

- *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.*
- *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*.

3.4.2 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 con soporte simultáneo para 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*.

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 es 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.

Otra importante referencia la constituye el Reusable Software Research Group (**RSRG**) de la Ohio State University[15]. Este grupo ha definido un pequeño conjunto de relaciones semánticas independientes de cualquier lenguaje entre componentes [Edwards et al., 1997], encuadrándose dentro del modelo de reutilización **3C**[16] [Tracz and Edwards,

[15] El **RSRG** es un grupo de investigación en reutilización del software constituido por unos 16 investigadores repartidos en diferentes Universidades de los Estados Unidos de América, cuyo centro de referencia es el Department of Computer and Information Science, en la Ohio State University.

[16] Este modelo establece un glosario de términos y una arquitectura recomendada para sistemas software soportados por componentes reutilizables. El nombre de este modelo (**3C**) proviene de las definiciones de Concepto, Contenido y Contexto.

- ***Concepto***: Es la abstracción capturada en un componente. Contiene qué es lo que hace el componente.
- ***Contenido***: Es la implementación de la abstracción. Contiene cómo el componente hace lo que hace.
- ***Contexto***: Es el entorno en el que el diseño del componente le permite operar. Contiene las restricciones que han de cumplirse en el entorno en el que el componente será reutilizado.

1989], [Tracz, 1990], [Edwards, 1990], [Weide et al., 1991], que sirve de marco de referencia conceptual para subsistemas software reutilizables. Las relaciones identificadas por el RSRG intentan describir las dependencias entre componentes a un nivel conceptual, de manera que las relaciones ofrezcan a los programadores y encargados del mantenimiento una información precisa sobre la forma en que los componentes pueden y deben ser usados. Del conjunto de relaciones definidas, las tres relaciones más generales son implementa, usa y extiende. A continuación se presenta una breve descripción de las tres[17]:

La relación principal es implementa. Describe la relación entre una implementación (*un elemento concreto*) y una especificación (*un elemento abstracto*). Informalmente se puede definir como "*Un elemento concreto X implementa un componente abstracto Y si y sólo si X exhibe el comportamiento especificado por Y*". La relación implementa expresa una dependencia del elemento X con respecto al elemento Y que ofrece una descripción abstracta de su comportamiento.

La relación usa describe una dependencia entre dos abstracciones, esto es, entre dos implementaciones, entre dos especificaciones o entre una implementación y una especificación. La última de estas tres opciones se define informalmente como sigue: "*Un elemento concreto X usa un componente abstracto Y si y sólo si X depende del comportamiento especificado por Y*".

La relación extiende expresa una dependencia entre dos elementos abstractos o entre dos elementos concretos. Informalmente se puede definir como: "*Un elemento X extiende un elemento Y si y sólo si toda la interfaz y el comportamiento descrito por Y está incluido en la interfaz y el comportamiento descrito por X*". La relación extiende no es una relación de herencia, ya que extiende representa una relación de comportamiento entre elementos software.

En los modelos de assets estudiados en este documento no se encuentran relaciones que representen enlaces entre elementos de diferente nivel de abstracción de forma explícita. Una excepción a este hecho se tiene en el modelo de asset definido por REBOOT, donde se 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

[17] Se recuerda que la definición de estas relaciones se realiza desde el punto de vista del trabajo del RSRG, para el cual existen elementos concretos (implementaciones) y elementos abstractos (especificaciones). Estas relaciones enlazan componentes que pueden ser abstractos (especificaciones) o concretos (implementaciones). Un componente abstracto describe un comportamiento funcional, qué servicios ofrece un subsistema. Un componente concreto describe una implementación, cómo son ofrecidos los servicios de un subsistema. Teniendo separados los componentes abstractos y concretos se soporta la abstracción de datos, ocultación de la información, múltiples implementaciones y descripciones autocontenidas del comportamiento de los componentes.

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, obteniéndose "*cosificar*" como una traducción poco formal de este verbo.

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[18] (*o jerarquía de modelos*) donde el metamodelado se basa en la idea de reificar las entidades que forman un cierto

[18] 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].

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, 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 embargo, referencias más adecuadas para justificar la consideración de la reificación como la relación estructural entre assets clasificados en diferentes niveles de abstracción se encuentran 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.

- Las relaciones semánticas entre elementos clasificados en diferentes niveles de abstracción constituyen el elemento estructural fundamental para soportar la noción de una estructura de reutilización compleja de grano grueso que soporte diferentes niveles de abstracción de forma simultánea.
- **ITHACA** vuelve a ser un punto de referencia válido para el estudio de las relaciones semánticas internivel para formar elementos reutilizables de grano grueso.
- Bajo las ideas del **RSGR** se puede apreciar claramente una intención de relacionar elementos clasificados en diferentes niveles de abstracción, aunque no se pretende en ningún momento definir un elemento multinivel de abstracción. Otro aspecto a destacar es que su modelo de reutilización está montado sobre su filosofía de implementación de sistemas software, donde las especificaciones no son otra cosa que tipos abstractos de datos, no cubriendo por tanto todo el ciclo de vida del software.
- Excepto en el modelo de asset propuesto por **REBOOT**, las relaciones entre assets clasificados en diferentes niveles de abstracción no se ven reflejadas en los modelos de assets propuestos por los principales repositorios.
- La reificación es la relación estructural más adecuada para representar el enlace entre assets clasificados en diferentes niveles de abstracción, considerándola como una relación que va enlazando los diferentes estados de abstracción por los que va pasando un elemento software desde su concepción hasta su implementación final; estados que se ven físicamente representados por los assets.

Cuadro 10. Consideraciones sobre las relaciones entre elementos de diferente nivel de abstracción.

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.

En el Cuadro 10 se recogen una serie de conclusiones sobre las relaciones entre assets clasificados en diferente nivel de abstracción.

4. Mecanos como elementos software 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., 1998a]:

- ***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 con soporte de diferentes niveles de abstracción de forma simultánea:*** La estructura debe acoger varios assets relacionados entre sí y clasificados en diferentes niveles de abstracción.
- ***Soporte para la trazabilidad:*** De forma que se pueda navegar por los diferentes assets a través de la red de enlaces existentes entre ellos, con especial interés en los enlaces entre los assets clasificados en diferentes niveles de abstracción.
- ***Definición de un soporte para el desarrollo para reutilización:*** La estructura debe dar soporte a los assets que se diseñen (*o se descubran con técnicas de reingeniería*) para su reutilización.
- ***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 son por tanto los elementos reutilizables de granularidad gruesa que ofrecen el soporte a diferentes niveles de abstracción de forma simultánea. Estos elementos reutilizables, como se ha mencionado anteriormente, deben integrarse en el proceso general de la reutilización del software[19] (*ver Figura 7*), esto es, deben estar

[19] En la Figura 7 se resume el proceso de reutilización según **REBOOT** [Karlsson, 1995]. Así, el desarrollo para reutilización conlleva todas las actividades encaminadas a preparar assets para que puedan ser reutilizados en otros contextos. El desarrollo para reutilización requiere la incorporación dentro del proceso de desarrollo general del software de unas actividades complementarias. Para un estudio más detallado de estas actividades se puede consultar [Karlsson, 1995], [García et al., 1997b], [Villa, 1997]. Por su parte, el desarrollo con reutilización hace referencia a la construcción de nuevos productos software haciendo uso de assets. Este proceso de desarrollo incluye cuatro actividades principales:

1. Búsqueda y recuperación un conjunto de assets candidatos.

2. Evaluación del conjunto de assets recuperados para encontrar los más adecuados.

3. Si es necesario, adaptación de los assets para ajustarlos a unos requisitos concretos.

4. Integración de los assets para formar el nuevo producto software.

soportados tanto por las actividades propias del desarrollo para reutilización como por las actividades propias del desarrollo con reutilización.

Esta dualidad en la creación de mecanos da lugar a un modelo de reutilización mixto composición-generación.

4.1 Requisitos de los mecanos

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., 1998a].

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 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 debe contar siempre con al menos una relación internivel. Esta situación da lugar a cuatro posibles configuraciones:

  1. **R-D-I**: Sería el caso normal. Se tienen assets definidos en los tres niveles de abstracción (*requisitos, diseño e implementación*) y relaciones internivel entre ellos.
  2. **R-D**: Un caso probable en el que se tienen especificaciones y diseños relacionados por relaciones internivel.
  3. **R-I**: Se tienen assets en los niveles de especificación e implementación, pero no en el nivel de diseño.
  4. **D-I**: Otro caso que 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.*** Cada uno de los assets componentes de un mecano debe pertenecer a un tipo de asset que recoja las características propias de ese tipo de elemento software y permita establecer un control sobre con que otros assets puede relacionarse.
- ***Se debe tener una flexibilidad para incrementar los tipos de assets soportados.*** El mundo del software es sumamente cambiante y evolutivo. Es frecuente la aparición de nuevos métodos de desarrollo software que utilizan nuevas técnicas, o variaciones de las existentes, para la generación de elementos software que pueden ser reutilizables.
- ***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.*** La información relacionada con un asset puede clasificarse en diferentes subsistemas de información. Por un lado están los atributos que capturan toda la información lógica del asset (*información derivada del tipo de asset al que pertenece, referente a sus creadores, sobre las personas que pueden acceder a él, las métricas y criterios de certificación…*) y por otro lado está la información que indica donde se encuentra ese asset físicamente (*fichero, URL…*).
- ***Un asset puede formar parte de varios.*** Un mismo asset puede formar parte de varios desarrollos (*eso al menos es el objetivo de la reutilización*), y por lo tanto puede aparecer como componente de distintos mecanos que conviven en el repositorio.
- ***Los enlaces semánticos entre assets deben ser de un tipo de relación:*** Cada uno de los enlaces entre los assets componentes de un mecano viene caracterizado por un tipo de relación semántica o relación de dominio.
- ***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 algunos de los tipos de relaciones semánticas que caracterizan los enlaces semánticos entre los assets coincidan en nombre con las relaciones estructurales necesarias para establecer el modelo.
- ***Los tipos de relaciones semánticas deben definir perfectamente las restricciones necesarias para evitar la introducción de incongruencias en la estructura de reutilización.*** Más concretamente, los tipos de 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 puede estar formado por un conjunto de mecanos existentes.*** Como se ha expresado anteriormente, un mecano es un agregado de assets relacionados. Pero, también puede definirse en función de otros mecanos.
- ***Un mecano está ligado a uno o a varios dominios.*** 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 está ligado a un contexto de reutilización.*** El contexto de un mecano 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 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 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 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.

5. Modelo de mecano

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 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: *el nivel de abstracción, la estructura del asset y las relaciones entre los assets componentes (inter e intranivel).*

5.1 Niveles de abstracción en un mecano

Un mecano soporta simultáneamente diferentes niveles de abstracción, niveles de abstracción que vienen marcados por las fases genéricas del ciclo de vida del software en las que fueron creados sus assets componentes.

Con esta decisión se está adoptando el criterio más extendido en la literatura por el cual se asocia 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 niveles de abstracción que comprende un mecano. Estando obligados a definir otro criterio de clasificación basado en algún otro criterio para la selección y recuperación de los assets componentes (*por ejemplo en facetas, donde el nivel de abstracción o fase de generación sea una de las facetas a considerar, pero no la única*).

Según lo expresado anteriormente, los mecanos pueden estar compuestos por un máximo de tres niveles de abstracción (*y un mínimo de dos*), que en orden decreciente de abstracción serían **nivel de requisitos**[20], **nivel de diseño** y **nivel de implementación**.

Esto conduce a contar con cuatro tipos de mecanos según su configuración de niveles de abstracción: **R-D-I**, **R-D**, **R-I**, y **D-I**.

5.2 Estructura de los assets componentes de un mecano (Asset GIRO)

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

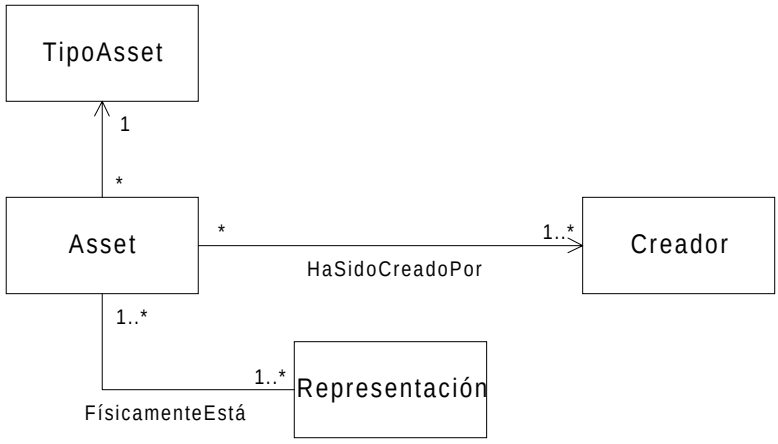

Figura 8. Modelo básico de asset.

[20] Otra alternativa válida, la cual se ajusta 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 pertenecen.

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 **TipoAsset, Representación** y **Creador**. En la Figura 8 se puede apreciar un primer modelo del núcleo básico de lo que será el modelo de asset GIRO[21].

El siguiente paso es el refinamiento de este modelo, 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(s):*** Área de aplicación, actividad o conocimiento del asset. La información del dominio es más propia del mecano al ser este un elemento reutilizable de mayor entidad que sus assets componentes. No obstante se ha pensado que un asset también puede almacenar la información de los dominios en los que puede ser reutilizado para no perder la flexibilidad de la posibilidad de recuperación de los assets componentes individualmente con independencia de la recuperación del mecano o mecanos con los que está asociado.
- ***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.** El valor de este atributo vendrá marcado por los niveles de abstracción que estén soportados por el tipo de asset al que pertenece.
- ***Método:*** Método de desarrollo utilizado para generar el asset.
- ***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 el asset.

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

- ***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.** El tipo del asset sirve para caracterizar al asset, así como para introducir un control en la forma en que los assets se relacionan. La clase **TipoAsset** tiene como atributos:

- ***Identificador:*** *Identificador único en el sistema.*
- ***Nombre:*** Nombre del tipo.
- ***NivelesAbstracción:*** Lista de los niveles de abstracción admitidos por el tipo.
- ***Paradigma:*** Paradigma de desarrollo bajo el que se ha generado.

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 el apartado siguiente.

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 8 podría refinarse y completarse como se muestra en la Figura 9.

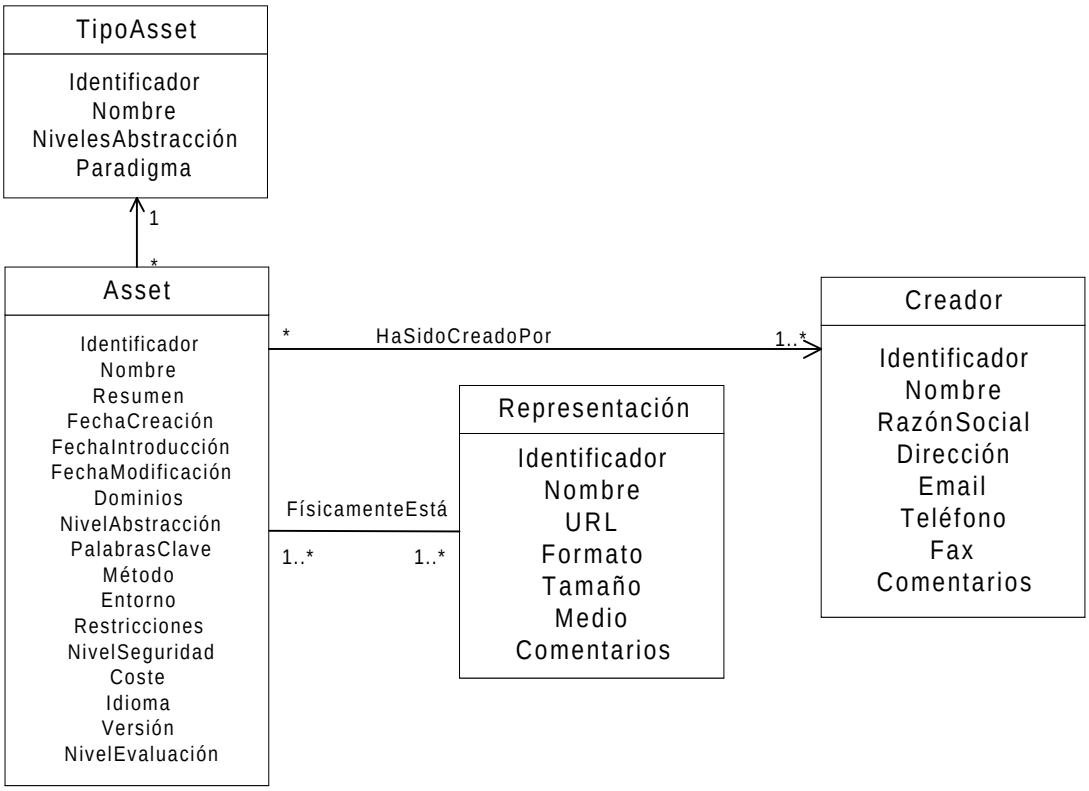

Figura 9. Modelo básico de asset GIRO.

5.3 Relaciones entre los assets componentes de un mecano

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.

Cada uno de los enlaces que se mantiene entre dos assets debe pertenecer a un tipo de relación semántica que recoja las restricciones características de la familia de enlaces a la que da lugar. Esto coincide con el concepto de power type [Martin and Odell, 1995].

Para capturar de una forma adecuada la semántica de los tipos de relaciones entre assets se ha decidido tomar como base de referencia un modelo objeto que aporte un conjunto

reducido de relaciones estructurales, que sirvan para caracterizar los tipos de relaciones semánticas entre los assets.

Las relaciones entre los assets componentes de un mecano están definidas en tres niveles de abstracción:

- En el nivel de abstracción más bajo están los enlaces entre los assets. Cada uno de estos enlaces entre assets pertenece a un tipo de relación semántica.
- Tanto los enlaces como los tipos de relación semántica (*a los que pertenecen los enlaces*) son instancias de las clases que en el modelo de mecano representan las relaciones (*clases* ***Relación*** *y* ***TipoRelación***), este nivel de modelo constituye el segundo nivel de abstracción en la definición de las relaciones entre assets.
- Pero a su vez cada uno de los tipos de relación semántica está soportado por una relación estructural definida en el modelo objeto, y que sirven para caracterizar los tipos de relaciones semánticas. Este es el tercer nivel de abstracción, correspondiéndose con un nivel meta.

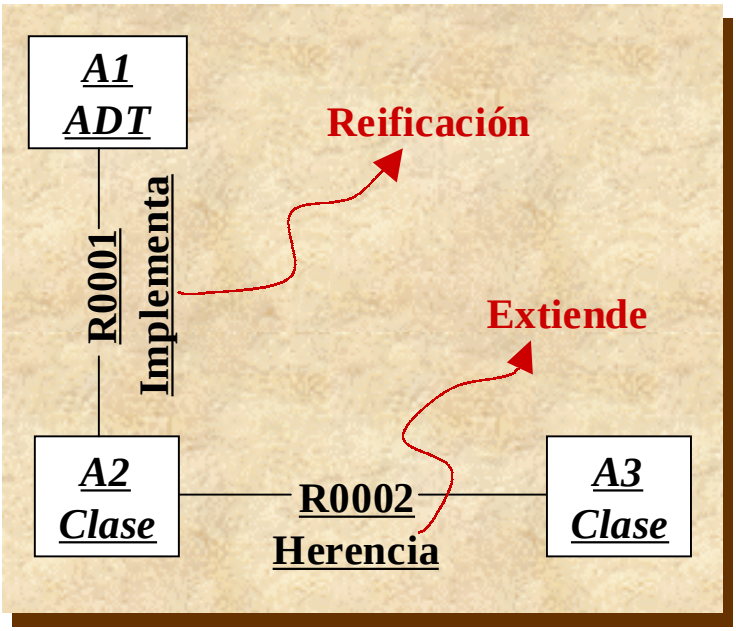

Figura 10. Diagrama de objetos que representa un mecano.

En la Figura 10 presenta un ejemplo de un mecano muy simple, en el que se pueden apreciar los tres niveles de abstracción en el que se definen las relaciones semánticas entre los assets componentes de un mecano. En el nivel de enlace (*nivel de abstracción más bajo*) se encuentran los enlaces semánticos **R0001** y **R0002** entre los assets **A1**-**A2** y **A2**-**A3** respectivamente[22]. Tanto **R0001** como **R0002** tienen un tipo de relación asociado (*Implementa y Herencia respectivamente*), pero a su vez cada uno de los tipos de relación semántica está soportado por una relación estructural, **Reificación** y **Extiende** en el ejemplo utilizado.

El ejemplo que ilustra la Figura 10 se utiliza en la Figura 11 para mostrar los diferentes niveles de abstracción utilizados para definir los mecanos como estructuras complejas de reutilización. El nivel de modelo presente en la Figura 11 se corresponde con una parte del modelo de mecano inicialmente presentado en [García et al., 1997a],

[22] Los assets A1, A2 y A3 tienen un tipo de asset (*ADT, Clase, Herencia respectivamente*), aunque este aspecto no es el que se está tratando en el presente apartado.

posteriormente refinado en [García et al., 1998a] y cuya versión final se presenta en el presente documento.

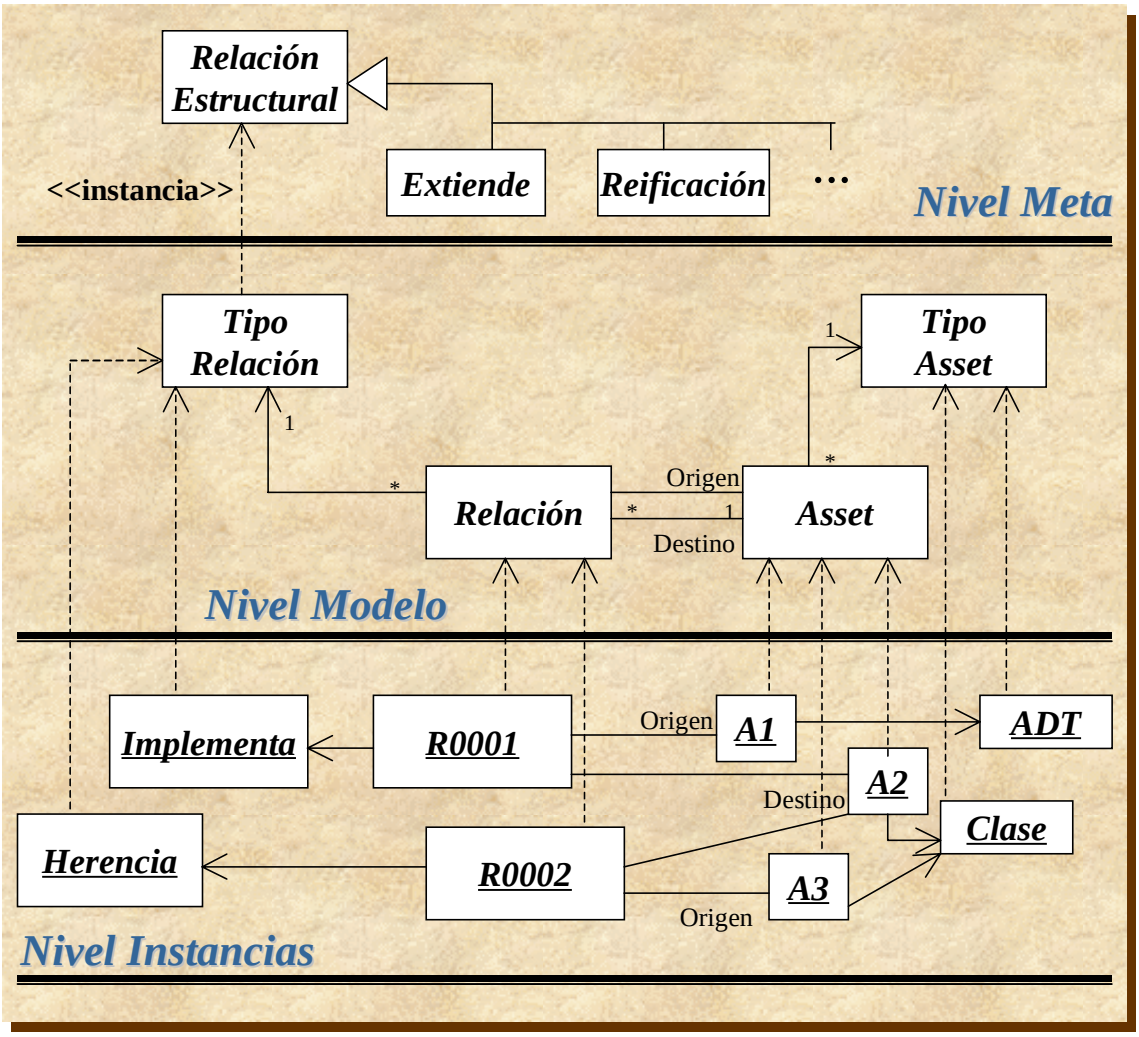

Figura 11. Niveles de abstracción en la definición de los mecanos.

Cada una de las relaciones estructurales a considerar debe concentrar en su definición unas connotaciones semánticas, que serán compartidas por todos los tipos de relaciones semánticas soportados por ellas, convirtiéndose así en la base constructiva y de soporte a la generación de los mecanos.

5.3.1 Definición de las relaciones estructurales

Uno de los aspectos más relevantes de los mecanos como estructura compleja de reutilización es su definición en diferentes niveles de abstracción simultáneamente, especialmente cuando se está haciendo referencia a los mecanos que se almacenan en el repositorio como elementos software reutilizables complejos. Por tanto, al existir siempre en un mecano assets clasificados en diferentes niveles de abstracción aparece la necesidad de diferenciar entre las relaciones que se dan entre assets clasificados en un mismo nivel de abstracción, relaciones intranivel, y las relaciones que enlazan assets clasificados en diferentes niveles de abstracción, relaciones internivel. Por consiguiente, es necesario que el modelo objeto subyacente aporte las relaciones estructurales adecuadas para dar soporte a ambos tipos de relaciones.

La primera relación estructural a considerar es la que viene impuesta por la restricción de que todo mecano debe contar con al menos dos assets clasificados en niveles de

abstracción diferentes y relacionados entre sí. Así, se necesita una relación estructural que soporte la semántica derivada del cambio de nivel de abstracción de un asset. Esta relación es la reificación, que expresa un refinamiento por el que se obtienen elementos software cada vez más concretos [Denker, 1996].

Desde el punto de vista del espacio de reutilización que se está construyendo, la reificación puede definirse como: "*Se dice que un asset Y reifica a un asset X cuando el asset Y se puede considerar como un refinamiento del asset X, y el asset X está clasificado en un nivel de mayor abstracción que el asset Y*".

Con la reificación se cubre el espectro estructural de las relaciones internivel. Pero, en lo que respecta a las relaciones intranivel se tiene una mayor diversidad semántica. En consecuencia en el espacio de reutilización se hace necesario un conjunto más amplio de relaciones estructurales que recojan las connotaciones semánticas siguientes: "ser parte de" o relaciones de meronimia, uso, extensión y asociación.

La semántica de inclusión se hace necesaria en el espacio de reutilización para representar la relación existente entre aquellos assets de granularidad gruesa que contienen un conjunto de assets de grano más fino. Las connotaciones semánticas que existen en torno a este tipo de relaciones de inclusión son muy variadas [Martin and Odell, 1995], [Henderson-Sellers, 1997]. No obstante, para los intereses de este trabajo se distinguen sólo dos tipos de relaciones estructurales de inclusión, la agregación y la composición. La agregación presenta una contención débil, donde las partes componentes tienen un ciclo de vida independiente del asset agregado, pudiendo pertenecer a varios assets agregados a la vez. Mientras que la composición implica una pertenencia más fuerte de las partes al asset compuesto, no teniendo sentido la reutilización de las partes por separado, ni su pertenencia a otro asset compuesto. Así pues, y desde el punto de vista del espacio de reutilización, la agregación y la composición se pueden definir como:

Agregación: *Se dice que un asset Y está relacionado con un asset X mediante una relación de agregación cuando el asset Y forma parte de la estructura de X, y además puede reutilizarse con independencia de la reutilización de X, estando X e Y clasificados en el mismo nivel de abstracción.*

Composición: *Se dice que un asset Y está relacionado con un asset X mediante una relación de composición cuando el asset Y forma parte exclusivamente de la estructura de X, no teniendo sentido la reutilización del asset Y sin la reutilización del asset X, estando X e Y clasificados en el mismo nivel de abstracción.*

La semántica de uso representa las relaciones tipo cliente/servidor donde un asset especifica un comportamiento (servicios) que otro asset cliente utiliza sin que el primero tenga que tener un conocimiento expreso de ello. Este tipo de comportamiento o de relaciones entre asset se representa por la relación estructural "usa a", que dentro del espacio de reutilización puede definirse como: "*Un asset Y usa un asset X cuando el asset Y depende del comportamiento especificado por el asset X para su definición o para completar su comportamiento, estando X e Y clasificados en el mismo nivel de abstracción*".

La semántica de extensión caracteriza todas aquellas relaciones en las que un asset se define en función de la definición ofrecida por otro asset. Este tipo de relaciones entre assets se representa por la relación estructural extensión, que dentro del espacio de reutilización se define como: "*Un asset Y extiende otro asset X cuando Y está definido en función de la definición de X, estando X e Y clasificados en el mismo nivel de abstracción*".

Las asociaciones se necesitan en el espacio de reutilización para expresar enlaces entre assets clasificados en el mismo nivel de abstracción, pero de manera que expresen los contenidos semánticos no soportados por los tipos de relaciones estructurales anteriores. La relación estructural asociación se define en el espacio de reutilización como: "*Un asset X está asociado con un asset Y cuando existe un enlace semántico bidireccional entre ambos assets, estando X e Y clasificados en el mismo nivel de abstracción*".

5.3.2 Restricciones de las relaciones

Una vez definidas las relaciones estructurales necesarias en el espacio de reutilización de definición de los mecanos, es necesario presentar una serie de restricciones que las caractericen. Estas propiedades son: el orden de las relaciones, la dirección de las relaciones y el carácter de las relaciones frente al proceso de reutilización.

- **Orden de las relaciones**

  Como puede deducirse de las definiciones de las relaciones estructurales enunciadas en el subapartado anterior, para el modelado de éstas se ha optado por la utilización de relaciones binarias. Aparte de la naturalidad y sencillez que aportan a los modelos las relaciones binarias, la decisión tomada se ve refrendada por los siguientes hechos:

  - Lo que interesa expresar con las relaciones entre assets es la trazabilidad de unos a otros. Esto es, se quiere expresar como un determinado asset se relaciona con otros assets (*independientemente del nivel de abstracción de éstos*), para que en el momento de su reutilización se esté en situación de reutilizar también todos aquellos assets relacionados con el primero[23].
  - El uso de relaciones semánticas binarias no conlleva pérdida semántica, ya que una relación de orden superior a dos siempre se puede modelar como un conjunto de relaciones binarias, aunque esto obligue a *reificar* alguna relación en alguna entidad (*en el modelado de datos*).
  - El uso de relaciones binarias es una práctica común en el modelado de sistemas, teniendo destacados ejemplos en el estándar BRM (Binary Relations Model) [Abrial, 1974], [Bracchi et al., 1976], [Girow, 1996] o en el estándar ODMG 2.0 [Cattell and Barry, 1997]. Además, este mismo criterio ha sido adoptado ya en reutilización, siendo ejemplos representativos de ello las relaciones entre componentes definidas por el RSRG [Edwards et al., 1997], o las relaciones entre los objetos soportadas por el Repositorio Microsoft [Microsoft, 1997], [Bernstein et al., 1997].

- **Dirección de las relaciones**

  Las relaciones estructurales identificadas en el espacio de reutilización se definen entre un origen y un destino, esto implica que aunque los enlaces semánticos entre los assets que forman los mecanos puedan ser recorridos en ambas direcciones, la

[23] Esto conduce a pensar en relaciones entre un asset y otros assets, de forma que aparecería una relación distinta por cada asset relacionado con él, esto es, relaciones binarias. Pero, si existiese un caso hipotético donde un método estableciera que un determinado producto software surgiera como relación de tres o más assets, tampoco existiría necesidad de una relación de orden superior a dos, porque el producto en sí mismo sería un asset (*si no lo fuera no sería algo que afectara a la reutilización*), y este asset de granularidad superior puede estar relacionado con sus componentes con relaciones binarias.

semántica expresada por la relación es sensitiva a la polaridad de la relación indicada por el origen y el destino.

En consecuencia todas las relaciones estructurales consideradas se tiene una polaridad semántica que se recoge en la Tabla 1, a excepción de la asociación, considerada bidireccional por definición al estilo de Booch [Booch, 1994] o de UML [Rational et al., 1997b].

Relación	Polaridad
Reificación	Del asset clasificado en el mayor nivel de abstracción al asset clasificado en el menor nivel de abstracción
Agregación	Del asset "todo" al asset "parte"
Composición	Del asset "todo" al asset "parte"
Uso	Del asset "cliente" al asset "servidor"
Extiende	Del asset "derivado" al asset "base"

Tabla 1. Relaciones estructurales con una polaridad semántica definida.

- **Carácter de las relaciones**

  Buscando la flexibilidad que permite el contar con la posibilidad de generar estructuras complejas de reutilización no fijadas a priori, se deben marcar cada uno de los enlaces semánticos entre assets de forma que se establezca su vinculación con respecto al proceso de reutilización, o más concretamente las dependencias entre las parejas de assets relacionados, aprovechando esta información para la generación de mecanos en tiempo de reutilización.

  Dado que las relaciones estructurales contempladas en el espacio de reutilización se pueden recorrer en ambas direcciones, dan lugar a dos roles que representan dos enlaces semánticos dirigidos en direcciones opuestas. A partir de esto, se define que dichos enlaces semánticos pueden ser fuertes o débiles. Se denomina enlace fuerte a aquél que cuando se rompe implica la pérdida del sentido de la reutilización del asset origen del enlace. Por el contrario, se denomina enlace débil a aquél que cuando se rompe, el asset origen del enlace puede seguir reutilizándose sin problemas. A continuación se presenta la caracterización de cada una de las relaciones estructurales consideradas con respecto al proceso de reutilización desde el punto de vista generativo que se está abordando.

  Si se aplican las definiciones de enlace fuerte y débil para determinar el carácter de los enlaces en la relación de reificación se tiene que ésta da lugar a dos enlaces semánticos débiles, pues un asset clasificado en un determinado nivel de abstracción puede reutilizarse independientemente de los assets con los que se relaciona en otros niveles de abstracción. Sin embargo, si se presenta el problema desde el punto de vista de la generación de mecanos, interesa cualificar al menos un enlace como fuerte, el dirigido desde el asset de mayor nivel de abstracción al asset de menor nivel de abstracción.

  Con la introducción de un enlace fuerte en la reificación se está asegurando la generación de mecanos en tiempo de reutilización como respuesta a las peticiones de los desarrolladores con reutilización, dado que bajo esta perspectiva generativa, la extracción del repositorio de un asset clasificado en un nivel de abstracción

superior al de implementación siempre se verá acompañada de la extracción de los assets clasificados en niveles de abstracción menores relacionados con él.

Esta aproximación establece un marco de trabajo general para el enfoque generativo soportado por el modelo de reutilización. No obstante, esta forma de actuación puede verse modificada por las restricciones introducidas en las peticiones de los usuarios.

La agregación representa una semántica de contención no excluyente, por tanto reutilizar el asset de grano grueso implica reutilizar los assets que lo forman, pero la reutilización de cualquiera de sus partes no conduce a la reutilización del asset agregado, dándose un enlace fuerte del asset de grano grueso a cada uno de los assets de grano fino y un enlace débil de cada asset de grano fino al asset de grano grueso.

R. Estruc.	Configuración		Tipo R. Semántica			Explicación
	Orig. ⇨ Dest.	Dest. ⇨ Orig.	Nombre	T.Ass.O	T.Ass.D	
Reificación	**Fuerte**	**Débil**	**Implementa**	**ADT**	**Clase C++**	Reutilizar el ADT implica extraer la clase C++ (*en el enfoque generativo de mecanos*), pero no viceversa
Agregación	**Fuerte**	**Débil**	**Incluye**	**Modelo Ambiental**	**Diagrama de Contexto**	Reutilizar el modelo de ambiente implica reutilizar el Diagrama de Contexto pero no viceversa
Composición	**Fuerte**	**Fuerte**	**SeComponeDe**	**Biblioteca de Funciones**	**Función**	Reutilizar una biblioteca de funciones supone reutilizar todas las funciones contenidas en ella, y reutilizar una función supone reutilizar la biblioteca que la contiene
Uso	**Fuerte**	**Débil**	**SeDefineEn**	**DFD**	**Dic. De Datos**	Reutilizar un DFD implica reutilizar el DD donde se definen sus flujos y almacenes, pero no viceversa
Extiende	**Fuerte**	**Débil**	**Realiza**	**Clase Concreta**	**Clase Abstracta**	Reutilizar la clase concreta requiere reutilizar la clase abstracta de la que se deriva, pero no al contrario
Asociación	**Débil**	**Débil**	**Versión**	**Clase**	**Clase**	Reutilizar una versión de una clase no implica reutilizar las versiones derivadas de ella ni sus ancestros

Tabla 2. Carácter de las relaciones estructurales en el proceso de generación de mecanos.

La composición implica enlaces fuertes en las dos direcciones debido a la dependencia por existencia de las partes con respecto al todo, como se indica en la definición.

La relación de uso presenta un enlace fuerte del cliente al servidor y un enlace débil del servidor al cliente, debido a que el cliente forzosamente tiene que conocer la existencia del servidor, pero esto no sucede en el caso opuesto.

La relación extiende presenta una dependencia del asset que extiende la definición con respecto al asset origen de la definición, por tanto la reutilización del asset derivado necesitará de la presencia del asset base que le proporciona la definición

pero no al contrario, estableciéndose un enlace fuerte del derivado al base y un enlace débil del asset base al derivado.

La asociación presenta siempre enlaces débiles en las dos direcciones de navegación, lo cual permite introducir una serie de tipos de relaciones semánticas muy importantes en el espacio de reutilización que, aunque no tienen una implicación directa en la generación de mecanos, son imprescindibles en la definición de elementos software reutilizables complejos, como por ejemplo el caso de la relación de versionado.

La Tabla 2 recoge para cada una de las relaciones estructurales del espacio de reutilización en el que se definen los mecanos, su configuración de enlaces con respecto al proceso de generación de mecanos, así como un ejemplo de un tipo de relación semántica que de ella se deriva, indicando los tipos de assets origen y destino de dicho tipo de relación semántica.

5.3.3 Relaciones en el modelo de mecano

Una vez que se ha estudiado cómo se definen las relaciones entre los componentes de un mecano, se puede proceder a modelar las relaciones dentro del modelo de mecano. En la Figura 11 aparece un esbozo de la parte del modelo de mecano referente a las relaciones entre los assets componentes, pero es en la Figura 12 donde esta parte queda totalmente reflejada.

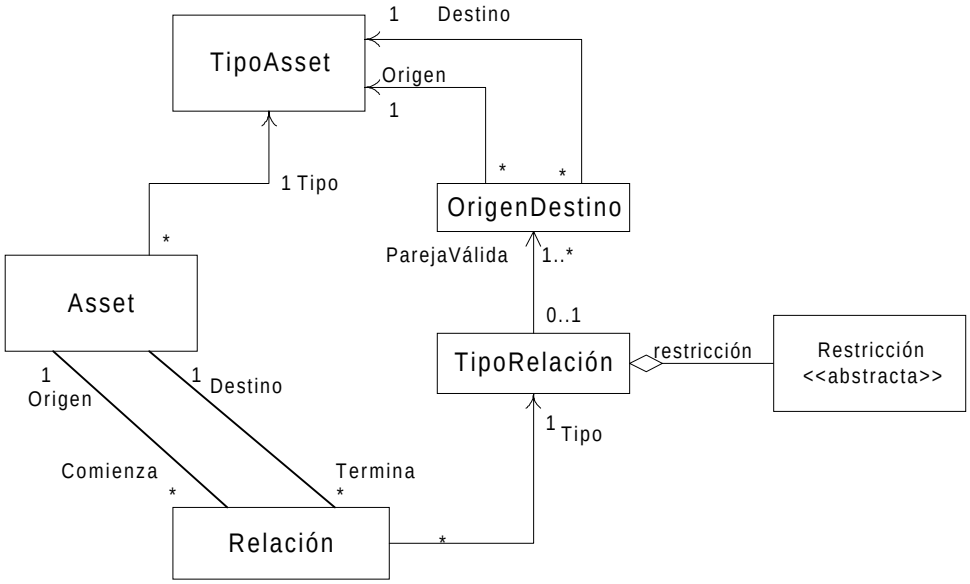

Figura 12. Modelado de las relaciones en el modelo de mecano.

La clase **Relación** es la responsable de establecer los enlaces entre los assets, es decir, es la responsable de crear los enlaces de menor nivel de abstracción. El único atributo necesario (*a parte de los necesarios para implementar las asociaciones de la clase*) es un nombre o identificador único para el enlace.

En cuenta a las asociaciones de la clase **Relación** se tiene que[24]:

[24] Utilizando **OCL** (*Object Constraint Language Specification*) [Rational et al., 1997d].

Relación

```
self.Origen   -- es de tipo   Asset
self.Destino  -- es de tipo   Asset
self.Tipo     -- es de tipo   TipoRelación
```

Cuando se establece un enlace entre dos assets se crea una instancia de la clase **Relación**, necesitando para ello sendas referencias a los assets que jugarán los roles de origen y destino de la relación, así como una referencia a uno de los tipos de relaciones semánticas existentes. Pero para que la creación de la instancia de la clase **Relación** sea efectiva se debe comprobar que se cumplen las restricciones oportunas para el tipo de relación semántica que se esté considerando.

Para el chequeo de las restricciones el método de construcción de la clase **Relación** delega en el objeto instancia de la clase **TipoRelación** cuya referencia ha recibido.

La clase **TipoRelación** contiene dos atributos:

```
Nombre:        String   -- Nombre único del tipo de relación semántica
EsInternivel:  Boolean  -- Bandera que indica si el tipo de relación es
                        -- o no internivel
```

La clase **TipoRelación** tiene una asociación con la clase **OrigenDestino** que en OCL se expresa:

TipoRelación

```
ParejaVálida  -- es de tipo   Set(OrigenDestino)
```

La clase **OrigenDestino** sirve para indicar que tipos de assets pueden estar relacionados, por lo tanto cada tipo de relación deberá contar con un conjunto de instancias de **OrigenDestino** que indique que tipos de assets pueden estar relacionados con ese tipo de relación.

Gracias a la asociación **ParejaVálida** se puede comprobar si el enlace que se va a realizar entre dos assets es congruente con la semántica establecida por el tipo de relación semántica que gobierna el enlace.

La responsabilidad de comprobar si una pareja de assets puede ser enlazada recae en **TipoRelación**, para lo cual debe existir un método cuya signatura sea:

```
TipoRelación::ComprobarPareja(_origen: Asset, _destino Asset): boolean
```

El método **ComprobarPareja** simplemente se limita a hacer la siguiente comprobación:

TipoRelación

```
self.ParejaVálida -> exits (_origen: Asset, _destino: Asset, _orgdes: OrigenDestino |
                            _orgdes.Origen.Nombre = _origen.Tipo.Nombre and
                            _orgdes.Destino.Nombre = _destino.Tipo.Nombre
                           )
```

La comprobación de que entre un asset origen y un asset destino se puede establecer un enlace es una comprobación propia del tipo de relación semántica, sin embargo existen otras restricciones que deben cumplirse y que vienen impuestas por la relación estructural que gobierna al tipo de relación semántica. Estas restricciones no pueden hacerse corresponder con responsabilidades de la clase **TipoRelación** porque cada tipo de relación, dependiendo de la relación estructural que marque su semántica, tiene unas restricciones diferentes.

Para implementar las restricciones de una forma adecuada se pueden seguir diferentes caminos, buscando con cualquiera de ellos un método que recoja las restricciones propias de cada tipo de relación estructural. Así, se podía haber considerado la clase **TipoRelación** como una clase abstracta y haber derivado tantas clases concretas como tipos de relación estructural se han considerado, de forma que cada una de ellas implemente sus propias restricciones. Otra posibilidad consistiría en utilizar de nuevo el concepto de **power type,** a semejanza de como se ha hecho con la clase **Relación** y la clase **TipoRelación**.

Pero la opción elegida ha sido aplicar el patrón **Strategy** [Gamma et al., 1995], de forma que se crea una familia de métodos para aplicar las restricciones propias de cada relación estructural. Para ello se cuenta con una clase abstracta denominada **Restricción** que definirá la interfaz del método de chequeo de restricciones, y después derivando por herencia la clase **Restricción** se obtiene una clase concreta para cada tipo de relación estructural considerado, de manera que cada clase implementa sus propias restricciones.

Para enlazar la clase **TipoRelación** con la clase **Relación** se ha utilizado una relación de agregación como se muestra a continuación:

```
TipoRelación
  self.restricción   -- es de tipo   Restricción
```

Como se ha dicho, cada una de las relaciones estructurales impondrá sus propias restricciones, aunque todas ellas deberán comenzar por comprobar si se cumple la condición propia de ser una relación internivel o una relación intranivel.

Para el caso de la relación de reificación la condición a cumplir es:

```
  not (self.origen.NivelAbstracción = self.destino.NivelAbstracción)
```

Siendo para el resto de las relaciones estructurales la condición:

```
    self.origen.NivelAbstracción = self.destino.NivelAbstracción
```

En la Figura 13 se presenta de nuevo la parte del modelo de mecano referente a las relaciones entre sus assets componentes, pero esta vez con un mayor grado de refinamiento en lo que se refiere a atributos y métodos.

5.4 Mecano como agregación de mecanos

Se ha venido definiendo un mecano como un sistema 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 puede contener a su vez otros mecanos, llegando a una agregación de mecanos.

La estructura definida en la Figura 14 puede permitir esta situación de forma que el mecano resultante fuera la unión de los assets que componen los mecanos que forman el nuevo mecano.

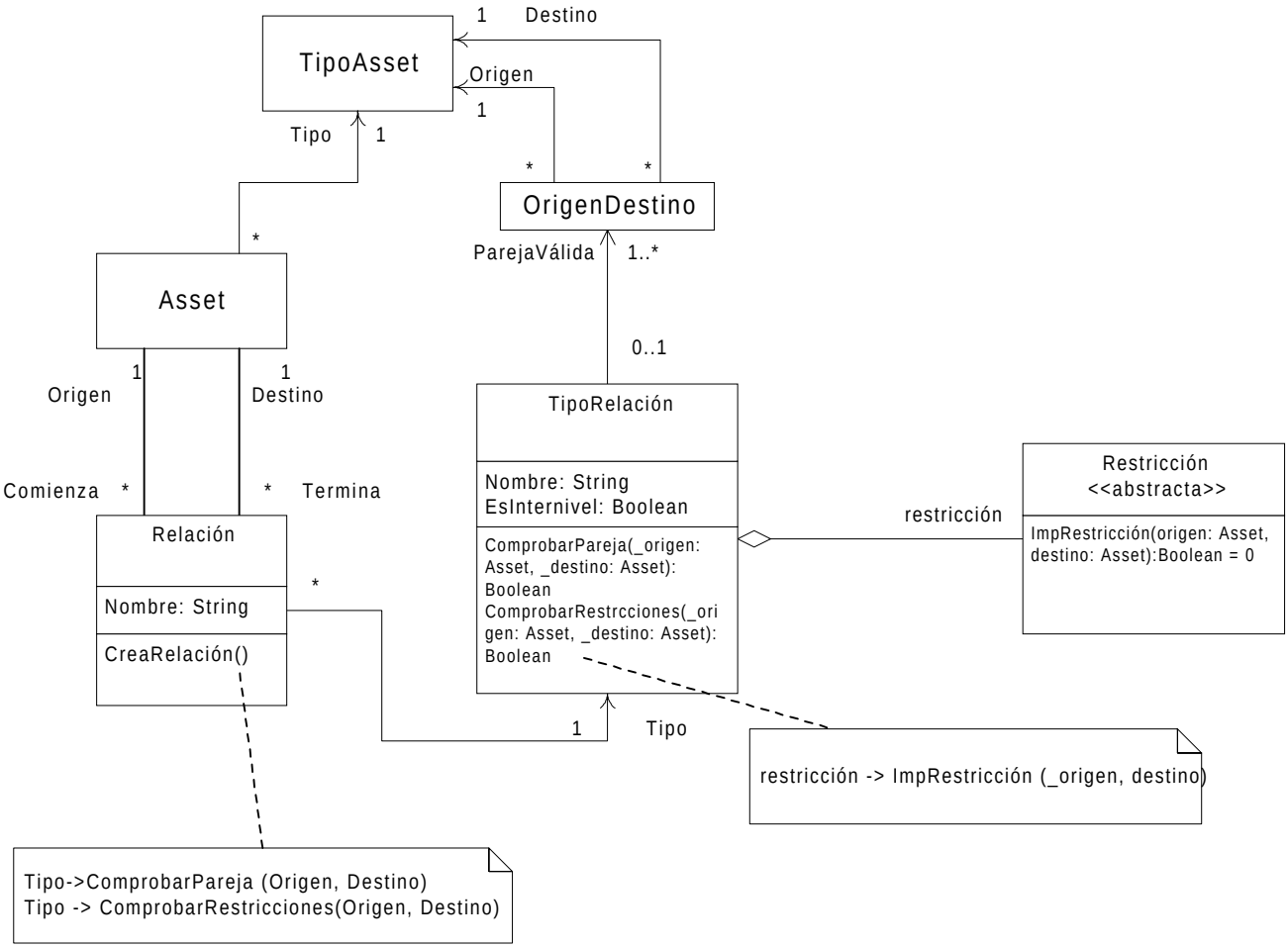

Figura 13. Modelo de las relaciones en un mecano con un mayor grado de detalle.

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

Se puede optar por otra solución que vendría de la mano de ampliar la definición de un mecano para incluir la posibilidad de construcción de mecanos mediante la agregación de éstos.

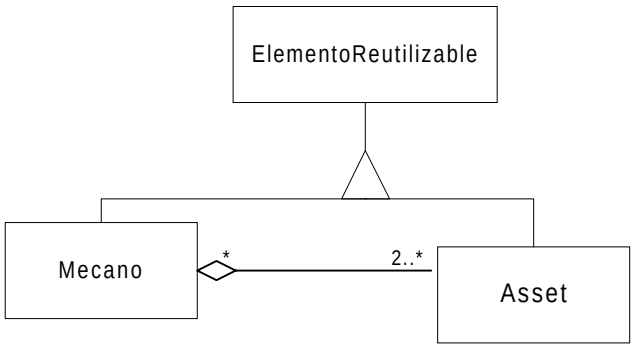

Figura 14. Modelo preliminar de un mecano.

5.5 Conclusiones sobre el modelo de mecano

Después del estudio realizado sobre la estructura de los mecanos, falta la unificación de todas las partes en lo que es el modelo de mecano que puede apreciarse en la Figura 15.

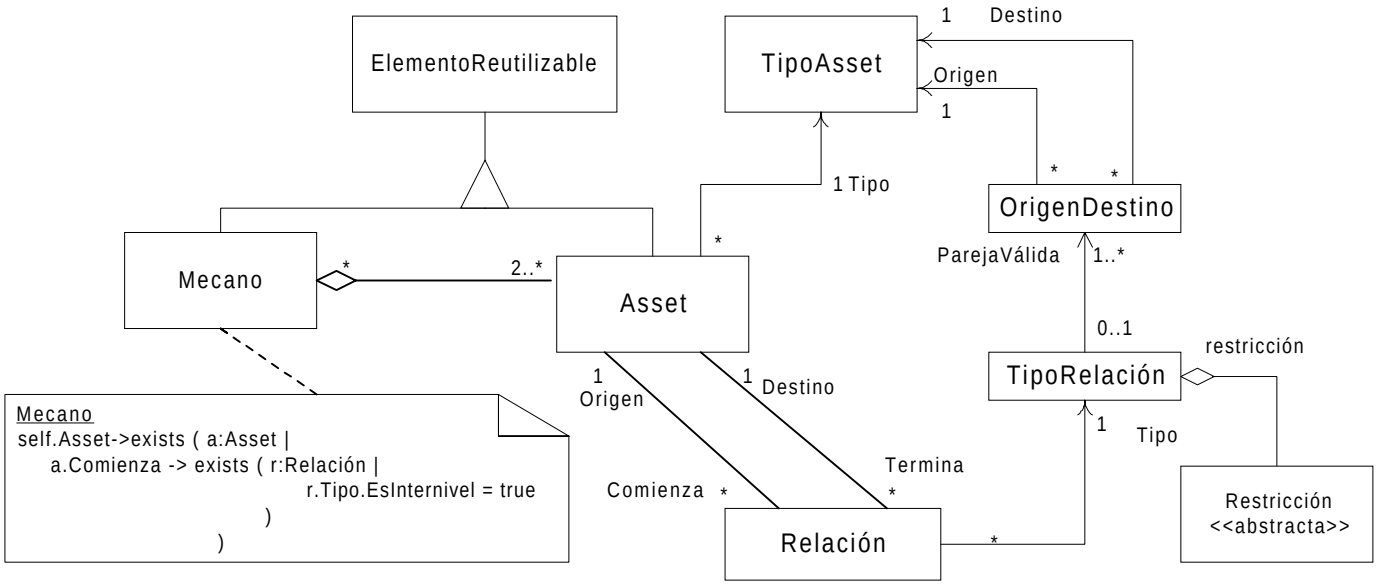

Figura 15. Modelo de mecano.

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

- *Se ha utilizado un modelo objeto para representar las entidades semánticas del problema, utilizando **UML 1.1** como lenguaje de modelado.*
- *En el modelo aparece la clase **Mecano**, que representa el elemento reutilizable complejo formado por una agregación de assets. Esta clase tiene como atributos un nombre y una lista de dominios en los que su reutilización es útil.*
- *Que un mecano sea una agregación y no una composición es fácil de argumentar. Cada uno de los componentes de un mecano puede ser parte de otros mecanos. Además, la eliminación de un mecano como elemento reutilizable no conlleva obligatoriamente a la eliminación de sus componentes.*
- *De los dos puntos anteriores se puede concluir que la connotación semántica de la relación de agregación entre el **Mecano** y los **Assets** se corresponde con lo que **UML 1.1** denomina agregación compartida, y que no es más que un tipo de agregación débil que implica que si se destruye el todo de la agregación (el mecano) no se tienen que eliminar las partes constituyentes (los assets), y lo que es más importante, las partes constituyentes pueden aparecer en varios agregados al mismo tiempo.*
- *El mecano presenta una restricción de definición importante que debe quedar reflejada en el modelo: 'todo mecano debe contar con al menos una*

*relación internivel'. Esta restricción aparece en el modelo como una sentencia **OCL** en una nota asociada a la clase semántica **Mecano**[25].*

- *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.*
- *Si un mecano se ve como el modelado conceptual de una estructura de información formada por assets y sus relaciones, el mecano está 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 se corresponde con los assets introducidos en el repositorio, siendo el nivel de modelado el modelo de mecano, habiéndose utilizado como lenguaje de modelado **UML 1.1**.*
- *El modelo de mecano presenta una flexibilidad total para la inclusión de nuevos tipos de assets y de relaciones semánticas. Sin embargo, el modelo está cerrado en cuanto al tipo de relaciones estructurales soportadas.*

6. Proceso de desarrollo con reutilización de mecanos

Si se parte de la hipótesis de la existencia de un repositorio poblado de una serie de mecanos, resultado de una serie de esfuerzos de desarrollo software para reutilización previos, sobre este repositorio se pueden llevar a cabo diferentes variantes de proceso de desarrollo con reutilización:

- **Reutilización de assets individuales**

  La reutilización de assets individuales, esto es, componentes de los mecanos almacenados en el repositorio, es una característica de obligado cumplimiento por cualquier repositorio, aunque se aleja del problema que aquí se está planteando y que está centrado en la reutilización de unos elementos software reutilizables de mayor nivel de granularidad, los mecanos. No obstante, la reutilización de un asset concreto puede verse como un caso especial de la reutilización de mecanos, de forma que cuando se ha seleccionado un determinado asset, éste se recupera sin

[25] Esta restricción podía haberse expresado mediante la siguiente sentencia **OCL**: `self.Asset.NivelAbstracción->asSet->size > 1`, pero se ha desestimado porque es mucho menos restrictiva que la que aparece en la Figura 15, ya que esta sólo hace referencia a que en un mecano todos sus componentes no pueden estar clasificados en el mismo nivel de abstracción pero expresa nada sobre que haya relaciones internivel. Sin embargo, la sentencia **OCL** de la Figura 15 se refiere a la necesidad de que exista al menos una relación internivel en un mecano, lo que cual ya implica la existencia de assets clasificados en al menos dos niveles de abstracción distintos.

más, sin necesidad de navegar a través de las relaciones que lo ponen en contacto con otros assets.

- **Reutilización de estructuras de grano grueso, pero que no constituyen un mecano**

  En el repositorio se pueden almacenar estructuras de grano grueso, es decir, agregaciones de assets, pero que no cumplen la condición necesaria para ser consideradas como mecanos, la existencia de relaciones internivel entre sus componentes.

  La recuperación de estas estructuras no es más que una variante de la recuperación de mecanos en los que no existen relaciones internivel.

- **Reutilización de mecanos**

  Si se parte del hecho de que un mecano es un elemento software reutilizable, éste tiene la propiedad de poder ser reutilizado. No obstante, el peso de este caso recae más en el proceso de creación del mecano, desarrollo para la reutilización por tanto, que en el proceso de desarrollo con reutilización.

  Si se limitase el proceso de desarrollo con reutilización a la recuperación de mecanos estáticos exclusivamente, se habrían conseguido dos objetivos importantes: *subir el nivel de abstracción de la reutilización* y *contar con un enfoque de reutilización multinivel*, pero se estaría perdiendo la potencia y flexibilidad que ofrece la combinación de diferentes mecanos de forma automática para generar un mecano que se ajustase a unos determinados requisitos establecidos en tiempo de reutilización y no en tiempo de creación.

- **Generación de mecanos en tiempo de reutilización**

  Este es el caso en el que converge tanto un esfuerzo de desarrollo para reutilización como un esfuerzo de desarrollo con reutilización, convirtiéndose así en el caso más interesante de estudiar.

  El proceso de generación de un mecano en tiempo de reutilización se basa en la siguiente idea: se han localizado un conjunto de assets (*por el procedimiento que implemente el gestor de repositorios utilizado*) pertenecientes a uno o a varios mecanos almacenados en el repositorio, procediéndose a la recuperación tanto de los assets seleccionados como de aquellos assets, relacionados con éstos, que sean necesarios para la óptima reutilización de los primeros.

  Bajo esta perspectiva se está formando un mecano en tiempo de reutilización que, en general, se puede pensar que va a estar compuesto por assets componentes de varios mecanos ya existentes (*mecanos persistentes*), siendo las relaciones entre los assets la clave para la generación de un mecano en tiempo de reutilización, ya que para obtener una estructura eficiente, el proceso de generación automático del mecano debe tener presente el conocimiento del proceso de desarrollo que guarda el mecano persistente embebido en las relaciones entre los assets componentes.

  El proceso de generación automática de mecanos en tiempo de reutilización se basa en la característica de las relaciones entre assets que las caracteriza frente al proceso de reutilización dando lugar a enlaces fuertes y débiles [García et al., 1998b].

  No obstante, existen una serie de factores que pueden obligar a ignorar el carácter de la relación justo en el momento de llevar a cabo el proceso de reutilización. Estos factores son:

- *La granularidad.*
- *La pertenencia a uno o varios mecanos.*

6.1 Factores que pueden obligar a ignorar el carácter de la relación

6.1.1 La granularidad

La granularidad hace referencia al grado de detalle o de descomposición con el que se consideran los assets. Un determinado asset puede ser descompuesto en sus elementos constituyentes para lograr un grado más fino de detalle a la hora de afrontar un problema, considerando a estos elementos constituyentes como assets también.

No obstante, por mucho detalle que se busque a la hora de representar ciertos assets, siempre existirá una frontera o límite físico que no podrá sortearse. Esta barrera física viene definida por la representación física del asset, que hará las veces de contenedor físico, y que en la mayoría de las ocasiones se corresponderá con un determinado fichero.

Este envoltorio físico en el que se encuentra almacenado el asset o los assets tiene unas implicaciones importantes en el proceso de reutilización. Reutilizar un asset físico implicará reutilizar todos los assets lógicos almacenados en dicho asset físico, y viceversa, si se desea reutilizar un asset incluido en otro asset de mayor granularidad (*que además es el que tiene una representación física*), se reutilizará tanto el asset buscado, como el asset de mayor granularidad, y por consiguiente también todos los assets componentes del asset de mayor granularidad.

Como consecuencia de esto, se puede afirmar que la granularidad mínima a la hora de reutilizar un asset vendrá dada por la representación física de los assets. Por tanto, un factor de suma importancia a la hora de diseñar los assets será determinar los assets lógicos que se almacenan en un determinado asset físico.

Como conclusiones sobre la influencia de la granularidad, se citan las siguientes:

- *Existen assets que cuentan con una representación física, y que serán candidatos para su reutilización independiente.*
- *Existen assets que no tienen una representación física propia, y que son introducidos como assets para bajar el nivel de granularidad de los elementos reutilizables y favorecer la selección de assets candidatos.*
- *Reutilizar un asset con representación física implicará reutilizar los assets lógicos en el incluidos, con independencia del carácter de las relaciones que vinculen a los assets de grano más fino al asset de grano grueso.*

6.1.2 La pertenencia del asset a uno o varios mecanos

Este es un aspecto de suma importancia porque va a suponer un criterio de poda a la hora de generar los mecanos de geometría variable.

Un enlace fuerte entre dos assets puede ser omitido en momento de reutilización debido a motivos del contexto de la reutilización marcados por relacionar assets pertenecientes a diferentes mecanos.

Dado que un asset puede pertenecer a varios mecanos, se puede ver el repositorio con una red de assets interconectados donde el nexo de unión entre mecanos vendría dada

por los assets que pertenecen a varios mecanos. Según esto, en un proceso de reutilización se obtiene un conjunto de assets relacionados, un mecano en tiempo de reutilización, que en un caso extremo podría llegar a ser igual a todo el contenido del repositorio, lo que a todas luces es del todo carente de sentido.

Esto impone la necesidad de marcar un límite al proceso de reutilización basado en enlaces fuertes. Cuando un asset está relacionado con otro asset por un enlace fuerte, pero este segundo asset no pertenece a un mecano incluido en el conjunto de mecanos involucrados en el proceso de reutilización en curso, entonces ese enlace fuerte no será considerado.

7. Conclusiones y trabajo futuro

Este trabajo recoge las bases de la definición no formal de un elemento software reutilizable caracterizado por su granularidad gruesa, su soporte simultáneo de diferentes niveles de abstracción, por su apoyo a la trazabilidad y flexibilidad para ser generado de forma automática con un enfoque compositivo/generativo mixto sobre una base de elementos reutilizables ya existentes. Este elemento software reutilizable recibe el nombre de mecano.

Como punto final del estudio de la estructura de un mecano se ha desarrollado un modelo que representa en núcleo central de éstos. Dentro de esta estructura las relaciones entre sus elementos componentes juegan un papel protagonista que tiene su máxima repercusión en el carácter de los enlaces entre los assets componentes frente al proceso de reutilización, carácter que va a permitir establecer un proceso automático para la generación de mecanos en tiempo de reutilización, y que va a dotar al entorno de reutilización de una flexibilidad y una de potencia que no se conseguirían de sólo contar con estructuras estáticas fijadas en tiempo de desarrollo para reutilización.

Como trabajo futuro quedan bastantes campos que abordar. En lo referente a la definición de los mecanos como elementos reutilizables queda determinar por completo las restricciones de las relaciones estructurales, así como identificar los tipos de assets y de los tipos de relaciones semánticas existentes entre ellos. No obstante, estas tareas no suponen ningún impedimento para la realización del trabajo en curso y pueden ser postergadas hasta el momento de la implementación de un repositorio que soporte el concepto de mecano utilizando un gestor de bases de datos objeto/relacional como puede ser **ORACLE 8**.

Un aspecto importante, relacionado tanto con el modelo de asset como con el modelo de mecano, es el establecimiento de una política de certificación y calidad para los assets y para los mecanos.

El estudio del versionado dentro de los mecanos es otro aspecto en el que se ha de trabajar para controlar la explosión combinatoria derivada de la introducción de versiones de los assets componentes de un mecano.

Además, una vez que se cuenta con la definición de un elemento reutilizable potente y flexible, como es el mecano, y con la experiencia de haber trabajado en desarrollos para reutilización, se está empezando a trabajar en el establecimiento de un modelo de reutilización con enfoque mixto compositivo/generativo basado en los mecanos. Este modelo de reutilización presentará tres vistas ortogonales: el modelo de técnico de reutilización, el modelo de proceso de reutilización y el modelo de calidad y métricas para la reutilización.

8. 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**.

9. 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**.**

[Bernstein and Dayal, 1994] Bernstein, Philip A., Dayal, Umeshwar. "*An Overview of Repository Technology*". In the proceedings of the 20th International Conference on Very Large Data Bases (Santiago – Chile. September 1994): 705-713. 1994.

[Bernstein et al., 1997] Bernstein P. A., Harry, B., Sanders, P., Shutt, D. and Zander J. "*The Microsoft Repository*". VLDB'97 Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece. Morgan Kaufmann, 1997.

[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.

[Creps et al., 1992] Creps, Richard E., Simos, Mark A. and Prieto-Díaz, Rubén. "*The STARS Conceptual Framework for Reuse Processes*". STARS'92. November 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, 1996] Cybulski, Jacob L. "*Introduction to Software Reuse*". Technical Report 96/4, Department of Information Systems, University of Melbourne (Australia). July 1996.

[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.

[Durnin et al., 1996] Durnin, Mary Anne, Terry, Kevin and Sullins, Rick. "*Establishing a Repository for Enterprise Wide Software Reuse*". In the proceedings of the Fifth Annual Workshop on Software Reuse Education and Training – Reuse'96 – (July 29, 1996 – August 1, 1996). http://www.asset.com/WSRD/conferences/proceedings/papers/sullins/mnpaper2.htm.1996.

[Edwards, 1990] Edwards, Stephen H. "*The 3C Model of Reusable Software Components*". In the proceedings of the Third Annual Workshop: Methods and Tools for Reuse. 1990.

[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.

[Gamma et al., 1995] Gamma, Erich, Helm, Richard, Johnson, Ralph and Vlissides, John. "*Design Patterns. Elements of Reusable Object-Oriented Software*". Addison-Wesley, 1995.

[García et al., 1997a] García, Francisco José, Marqués, José Manuel y Maudes, Jesús Manuel. "*Mecano: Una Propuesta de Componente Software Reutilizable*". En las actas de las 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 DiseÂOt.*

[García et al., 1998a] 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.

[García et al., 1998b] García Peñalvo, Francisco José, Marqués Corral, José Manuel, Laguna, Miguel Ángel y Maudes Raedo, Jesús Manuel. "*Influencia de las Relaciones entre Elementos Software Reutilizables en la Generación de Mecanos*". Aceptado en las III Jornadas de Ingeniería del software (Murcia, Noviembre de 1998). 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.

[Henderson-Sellers, 1997] Henderson-Sellers, Brian. "*OPEN Relarionships – Compositions and Containments*". Journal of Object-Oriented Programming (JOOP), pp. 51-55. November/December, 1997.

[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.

[Kara, 1997] Kara, Dan. "*The Repository's Role in Component Development*". http://cool.sterling.com/cbd/kara.htm. 1997.

[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.

[López, 1997] López Tallón, Alberto. "*COM y DCOM o Microsoft Contraataca*". Revista Profesional para Programadores (RPP), Nº31: 22-28. Julio, 1997.

[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.

[Marqués, 1998] Marqués Corral, José Manuel. "*Soporte Operativo para la Reutilización del Software: Repositorios y Clasificación*". En las actas del curso Ingeniería de Software y Reutilización: Aspectos Dinámicos y Generación Automática (Ourense, del 6 al 10 de Julio de 1998). 1998.

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

[Martin and Odell, 1995] Martin, J. and Odell, J. J. "*Object-Oriented Methods: A Foundation*". Prentice-Hall, 1995.

[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.

[Microsoft, 1997] Microsoft. "*Microsoft Repository Documentation*". Microsoft Corporation. April, 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.

[NATO, 1992] NATO. "*NATO Standard for Management of a Reusable Software Component Library*". Volume 2 (of 3 Documents) NATO Communications and Information Systems Agency. 1992

[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.

[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.

[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.

[Rational et al., 1997d] 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. "*Object Constraint Language Specification*". 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.

[Rogerson, 1997] Rogerson, Dale. "*Inside COM*". Microsoft Press. 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.

[Solderitsch, 1992] Solderitsch, James. "*Making the Case for Interoperating Reuse Libraries*". In the proceedings of 5th Workshop on Institutionalizing Software Reuse (WISR5) (Palo Alto, California). 1992.

[Solderitsch et al., 1992] Solderitsch, James, Thalhamer, John and Creps, Dick. "*Asset Library Open Architecture Framework – Sharing Reusable Assets*". In the proceedings of DARPA Software Technology Conference 1992 (Los Angeles, California – April 1992): 242 – 249. 1992.

[Szyperski and Pfister, 1996] Szyperski, Clements and Pfister, Cuno."*First International Workshop on Component-Oriented Programming WCOP'96*". 8 July 1996.

[Tracz, 1990] Tracz, W. "*Where Does Reuse Start?*". In the proceedings of the Realities of Reuse Workshop, Syracuse University CASE Center. January, 1990.

[Tracz and Edwards, 1989] Tracz, W. J. and Edwards, Stephen H. "*Implementation Working Group Report*". In the proceedings of the Reuse in Practice Workshop, Pittsburgh, PA, 1989.

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

[Viasoft, 1997] Viasoft Inc. "*The Rochade Repository Environment. The Foundation for the Information Asset Warehouse*". Viasoft Incorporation White Paper. http://www.viasoft.com/rochade/whitepaper/. October, 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.

[Weide et al., 1991] Weide, Bruce W., Ogden, William F. and Zweben, Stuart H. "*Reusable Software Components*". In M.C. Yovits editor, *Advances in Computers*., volume 33, pp. 1-65. Academic Press. 1991.