FRANCISCO JOSÉ GARCÍA PEÑALVO

MODELO DE REUTILIZACIÓN SOPORTADO POR ESTRUCTURAS COMPLEJAS DE REUTILIZACIÓN DENOMINADAS MECANO

EDICIONES UNIVERSIDAD DE SALAMANCA

COLECCIÓN VITOR
53
c
Ediciones Universidad de Salamanca
y Francisco José García Peñalvo

1ª edición: Julio, 2000
I.S.B.N. : 84-7800-920-5
Depósito Legal: S. 681-2000
Ediciones Universidad de Salamanca
Apartado postal 325
E-37080 Salamanca (España)

Realización:
Nemática, S.L.

Impreso en España – Printed in Spain

K

CEP. Servicio de Bibliotecas

GARCÍA PEÑALVO, Francisco José

Modelo de reutilización soportado por estructuras complejas de reutilización denominadas mecanos [Archivo de ordenador] / Francisco José García Peñalvo.-
Salamanca : Ediciones Universidad de Salamanca, 2000
1 disco compacto.-- (Colección Vítor ; 53)
Tesis-Universidad de Salamanca

1. Universidad de Salamanca (España)- Tesis y disertaciones académicas.
2. Programación de ordenadores-Reutilización.

681.3.068 (043.2)

A Mary Cruz y a mis padres

Resumen

La incorporación de un enfoque sistemático de reutilización en el desarrollo del software es una estrategia a seguir para aumentar la productividad y la calidad de los productos software.

El trabajo realizado se centra en la definición de un modelo de reutilización sistemática fundamentado en una estructura de elemento software reutilizable de grano grueso que va a denominarse *mecano*.

Un mecano se define como un conjunto de elementos software reutilizables de grano fino, clasificados en diferentes niveles de abstracción y relacionados entre sí, ya sea dentro del un mismo nivel de abstracción o entre diferentes niveles de abstracción, cumpliéndose la restricción de que debe existir al menos una relación internivel.

Este modelo de reutilización está formado por tres submodelos: *el modelo técnico de reutilización, el modelo de proceso de reutilización* y *el modelo de cualificación de reutilización.*

El modelo técnico de reutilización se centra en todos los elementos tecnológicos, herramientas y actividades necesarios para la definición y creación de mecanos, tanto en el desarrollo para reutilización como en el desarrollo con reutilización. Planteándose un enfoque híbrido composición/generación en dichos procesos.

El núcleo de esta vertiente técnica está formado por el modelo de componente reutilizable, definido tanto desde una perspectiva semi-formal utilizando técnicas diagramáticas propias del lenguaje de modelado UML (*Unified Modeling Language*) completadas con sentencias OCL (*Object Constraint Language*), acompañado en este caso del modelo de repositorio necesario para su soporte, como desde un punto de vista formal recurriendo al grafo tubo como herramienta formal sobre la que definir el concepto de mecano bien formado, además de una gramática de grafos dependiente del contexto para mecanos bien formados.

Conjugar los dos enfoques clásicos de reutilización, la composición y la generación, dota de una mayor flexibilidad a la propuesta de reutilización presentada, especialmente porque la parte de generación es completamente automatizable gracias a definición formal

del componente reutilizable.

La generación tal y como se concibe en este trabajo no se ajusta al tratamiento tradicional que de este enfoque se da en la bibliografía especializada, sino que se entiende como una construcción automática de mecanos por recolección de sus componentes de grano fino, gracias a las mallas que forman éstos y sus interrelaciones en el repositorio que los acoge.

El modelo de proceso de reutilización es el encargado de la organización de todas las actividades, recursos y procedimientos relacionados con la política de reutilización de un organismo.

Una de las partes más relevantes del modelo de proceso es la que se encarga de la definición de los procesos de creación de mecanos, que vienen diferenciados por un enfoque estrictamente composicional en el desarrollo para reutilización, donde los mecanos son compuestos *manualmente* por un desarrollador para reutilización; y por un enfoque híbrido composición/generación automático cuando, en el desarrollo con reutilización, los mecanos existentes no satisfacen las necesidades del desarrollador con reutilización.

El modelo de cualificación permite informar al desarrollador con reutilización sobre la calidad de los elementos que va a utilizar, estableciéndose un plan de calidad, un plan de métricas y un esquema de auditoría.

El plan de calidad está centrado en los factores de reusabilidad y fiabilidad de los componentes de los mecanos. El plan de métricas se deriva del propio plan de calidad y establece el conjunto de métricas que permite la obtención de medidas tanto de los productos manejados (elementos reutilizables de grano fino y de grano grueso) como del propio proceso de reutilización. Por último, el esquema de auditoría posibilita la evaluación independiente de los productos y procesos software, conforme a certificar objetivamente el cumplimiento del conjunto de principios y reglas establecidos en el modelo de cualificación.

Completando las propuestas teóricas que conforman el presente trabajo se han llevado a cabo diversas experiencias prácticas para obtener entornos de reutilización experimentales donde implementar el concepto de mecano.

Descriptores: Reutilización sistemática del software; elemento software reutilizable de grano grueso; modelo de reutilización sistemática del software; proceso de desarrollo de software con reutilización; cualificación de elementos reutilizables.

Abstract

Systematic software reuse activities should take place to avoid the *ad hoc* reuse approaches, thus producing an increase in productivity and a better quality of the software products.

The work is about the definition of a systematic software reuse-model based on a coarse-grained software element structure called *mecano*.

A mecano could be defined as a set of fine-grained reusable software elements that are classified in different abstraction levels and related to each other, both in the same and in different abstraction levels, where at least an interlevel relationship must exist as the only constraint.

This reuse model consists of three submodels: the technical reuse-model, the process reuse-model and the qualification reuse-model.

The technical reuse-model deals with all the activities related to the management of the mecanos, both in the development for reuse and in the development with reuse. It presents a compositional/generative duality in the creation of mecanos, being a hybrid approach to systematic reuse.

The core the technical part of the reuse model is the coarse-grained reusable component model. It is defined both in a semi-formal way, using UML (*Unified Modeling Language*) diagrammatic techniques in addition with OCL (*Object Constraint Language*) sentences, and in a formal way using tube graphs as formal tool to model the mecanos and also to define the well-formed mecano concept. A graph grammar for well-formed mecanos is defined too.

The semi-formal mecano model includes the repository model for these reusable elements support.

The conjugation of both classical reuse approaches, composition and generation, gives a bigger flexibility to the presented reuse proposal, especially because the generation approach can be completely automatic due the reuse component formal definition.

The meaning of the generation in this work isn't the same as it appears in the traditional literature. Here it is presented as an automatic construction of mecanos collecting

its fine-grained software components through the networks formed by these and their interrelationships in the repository.

The process reuse-model covers the logical organization of the personal, resources, methods, procedures, etc. into work activities designed to produce a specific end result, a work product.

One of the most important issues of the process model is the life-cycle reuse processes definition to operate with mecanos. We can differentiate both a strict compositional approach in the development for reuse, where a developer for reuse composes the mecanos manually and an automatic hybrid compositional/generative approach when the existing mecanos do not satisfy the developer with reuse's necessities.

The qualification reuse-model informs the developer with reuse about the quality of the reusable elements, which are going to be reused. It establishes a quality plan, a metrics plan and an audit scheme.

The quality plan is centred in the reusability and reliability factors of the components of one mecano. The metrics plan is derived from the quality plan and it establishes the set of metrics that allows obtaining the measures of both products (reusable elements: assets and mecanos) and the reuse process. Lastly, the audit scheme facilitates the independent evaluation of the products and the process to certify the fulfilment of the principles and rules established by the qualification model.

Completing the theoretical proposals of this work, different practical experiences have been developed to implement the mecano concept.

Keywords: Systematic software reuse; asset; coarse-grained reusable element; systematic software reuse model; process reuse-model; qualification reuse-model.

Índice General

Capítulo 1

Introducción

Ante una demanda voraz de productos software por una creciente parte de la sociedad, la producción económica de sistemas software de calidad es uno de los retos más serios a los que se enfrenta la Ingeniería del Software en la actualidad. Este hecho incide en que los mayores esfuerzos en investigación y desarrollo se centren en los métodos, técnicas y herramientas que mejoren el desarrollo eficiente de los sistemas software.

Se han producido grandes anuncios acerca de la "*solución definitiva*" a todos los problemas de la Ingeniería del Software, lo que se ha dado en llamar las "*balas de plata*" de la Ingeniería del Software en la literatura especializada [Brooks, 1987], entre las que cabe citar entre otras a los lenguajes de cuarta generación, la tecnología CASE o el paradigma objetual. Sin embargo, ninguna de ellas ha terminado por cuajar como la verdadera piedra filosofal que acabara con todos los males del software, convirtiéndose por contra en *grandes losas* con las que deben cargar los nuevos entornos de Ingeniería del Software que intentan combinar lo mejor de cada una de dichas aproximaciones.

En este sentido, la reutilización del software es considerada por muchos autores como uno de los enfoques más adecuados para incrementar la productividad, ahorrar tiempo y reducir los costes de los desarrollos de software [Biggerstaff, 1992], [Frakes and Isoda, 1994], [Karlsson, 1995], [Jacobson et al., 1997], [McClure, 1997], frente al absurdo, traducido en pérdida de productividad, competitividad y calidad, que supone abordar todas las aplicaciones desde cero, convirtiéndose así la reutilización del software en uno de los principales temas de investigación en el campo de la Ingeniería del Software. Más concretamente, Mili et al. [Mili et al., 1995] llegan a afirmar que "*La investigación en estas décadas en los campos de la Ingeniería del Software y de la Inteligencia Artificial ha dejado algunas alternativas, pero la reutilización es la 'única' aproximación realista para llegar a los índices*

de productividad y calidad que la industria del software necesita".

No obstante, pese a estas afirmaciones tan categóricas, las experiencias reales de la adopción sistemática de la reutilización en los procesos de construcción del software son escasas, aunque existentes[1], debido a la enorme complejidad que conlleva la concepción y adopción de un plan de reutilización por una organización.

Desde la perspectiva del presente trabajo de tesis doctoral, se pretende establecer un marco de referencia para la introducción de la reutilización sistemática dentro de los ambientes de desarrollo de software, que surgen como fruto de la adopción conjunta de diversas vertientes dentro de la Ingeniería del Software.

Este primer capítulo se organiza como sigue: la primera sección se dedica a realizar una introducción a la reutilización sistemática del software, presentando los términos y conceptos que se van a manejar a lo largo de este trabajo. La segunda sección recoge los problemas que sufren la reutilización del software, poniendo trabas para que esta disciplina madure y sea una realidad en el mundo del desarrollo del software. La tercera sección presenta los objetivos con los que se inició el presente trabajo. La sección cuatro explica someramente el contexto en el que se fraguó este trabajo de tesis doctoral, así como los objetivos que se esperan lograr. Por último, la quinta sección sirve como esquema general o resumen de los contenidos del resto de los capítulos.

[1] No se tienen excesivas muestras de casos donde la implantación de la reutilización haya dado resultados exitosos. Es de destacar en este aspecto las experiencias llevadas a cabo por Ivar Jacobson en **Ericsson** para la implementación del sistema de telefonía AXE, o el programa de reutilización adoptado por **Hewlett-Packard** bajo la supervisión de Martin Griss. Para una mayor información sobre estas experiencias se recomienda la lectura de [Jacobson et al., 1997]. Otra importante fuente de experiencias prácticas en la adopción de la reutilización del software se tiene en los programas **PIE** (*Process Improvement Experiment*) y **PIER** (*Process Improvement Experiment in Reuse*) financiados por el **ESSI** (*European Systems & Software Initiative*). Entre las numerosas iniciativas promovidas por el ESSI se pueden destacar **ROADS** (*Reuse Oriented Approach for Domain based Software*) dedicado a establecer un marco para la adopción de la reutilización, estando soportado por el **ESI** (*European Software Institute*) y por **PHS** (*Prosperity Heights Software*), llevando a cabo sus experiencias prácticas en **Thomson-CSF**, para una mayor información se puede consultar [Bandinelli and Rementería, 1996] y [Bandinelli and Sanz, 1997]; **SURF** (*Software re-Use: a process impRovement experiment at an IBM Italia Facility*) que tiene como objetivo una aproximación al software de calidad mediante la introducción de la reutilización [Riva, 1998a], [Riva, 1998b]; **GEARS** (*Gaining Efficiency and quAlity in Real time control Software*) que establece un modelo de reutilización orientado en componentes para la mejora del proceso de desarrollo de software de tiempo real [Carbone and Araneo, 1999].

1.1 Reutilización Sistemática del Software

La reutilización es un elemento central de la Ingeniería del Software moderna, con impacto en el resto de los campos de esta disciplina (*métodos formales, orientación a objetos, arquitecturas, procesos...*).

La reutilización se basa en una idea simple, aunque bien conocida. Se trata de utilizar elementos software previamente desarrollados a la hora de acometer la construcción de un nuevo producto software. Sin embargo, algo tan simple a nivel conceptual es difícil de llevar a la práctica, porque los elementos a reutilizar tienen que haber sido cuidadosamente diseñados, desarrollados y documentados para ofrecer un alto nivel de calidad al trabajar juntos. Esto se traduce en una mayor inversión inicial, que sólo se verá sufragada con la reutilización efectiva de estos elementos varias veces a medio/largo plazo.

El establecimiento de un proceso de desarrollo de software basado en la reutilización de elementos software diseñados para este fin, es un síntoma de madurez propio de disciplinas ingenieriles con mayor tradición y experiencia que la ingeniería informática.

Con el objetivo de fijar el contexto y el vocabulario en el que se va a encuadrar esta tesis, se va a proceder a realizar un somero repaso por los aspectos básicos de este área.

1.1.1 Definición de Reutilización del Software

Desde que, en la ya mítica conferencia sobre Ingeniería del Software de la **OTAN** en 1968 [Buxton et al., 1976], Doug McIlroy acuñara el término reutilización del software al proponer la idea de bibliotecas compartidas de componentes de código [McIlroy, 1976], se han producido diferentes definiciones de las que cabe destacar, por su trascendencia en la bibliografía especializada en este campo de la Ingeniería del Software, las siguientes:

1. *Cualquier procedimiento que produce (*o ayuda a producir*) un sistema mediante el nuevo uso de algún elemento procedente de un esfuerzo de desarrollo anterior [Freeman, 1987a].*

2. *Es el uso en una nueva situación de conceptos u objetos previamente adquiridos, lo cual implica la codificación de la información de desarrollo en diferentes niveles de abstracción, el almacenamiento de esta representación para futuras referencias, la comparación de situaciones nuevas y antiguas, la duplicación de acciones y objetos ya desarrollados y su adaptación para soportar nuevos requisitos [Prieto-Díaz, 1989].*

3. *Es el proceso de utilizar elementos software existentes en lugar de construirlos desde el principio. Típicamente, la reutilización implica la selección, especialización e integración de dichos elementos software, aunque diferentes técnicas de reutilización pueden enfatizar o quitar importancia a dichas fases [Krueger, 1992].*

4. *Es el proceso de implementar o actualizar sistemas software utilizando* ***assets*** *software ya existentes [Peterson, 1991], [DoD, 1996].*

Haciendo referencia a un área menos oportunista y más madura surge el concepto de **reutilización sistemática del software** que se define como "*una aproximación de carácter institucional para producir desarrollos en los que los assets son intencionadamente creados o adquiridos para ser reutilizables [Griss, 1993a], [Griss, 1995d], [Griss, 1996c]*".

Un concepto relacionado con el de reutilización del software, pero que no debe confundirse con éste, es el de *reusabilidad de un elemento software*, y que puede definirse como "*la medida de la facilidad con la que pueden utilizarse conceptos o elementos software previos en nuevas situaciones [Prieto-Díaz, 1989]*".

1.1.2 Elementos Software Reutilizables

Inicialmente el concepto de reutilización se vincula casi exclusivamente al proceso de reutilización de código fuente, bien de una forma *ad hoc*, bien de una forma más organizada construyendo bibliotecas de funciones. Sin embargo, el objetivo de la reutilización es ampliar su espectro de actuación a cualquier producto y/o conocimiento derivado de la producción de software [Freeman, 1987b].

Desde esta perspectiva, un elemento software reutilizable puede hacer referencia a cualquier producto software obtenido en el ciclo de vida del software, con independencia de su nivel de abstracción. Esta circunstancia provoca la aparición de una serie de términos en la bibliografía para referenciar este nuevo concepto (*artefacto software reutilizable, componente reutilizable...*), pero sin duda alguna, desde que Michael J. Lyon hablara de los "*very valuables assets*" para hacer referencia a los almacenes de software existentes [Lyon, 1981], el término **asset**[2] se ha adoptado como el término estándar en el campo de la reutilización para hacer referencia a los elementos software reutilizables.

En principio la utilización del término **asset** puede resultar extraña para la mayoría

[2]El término asset se traduce al español como valor o joya, traducción que puede dar lugar a equívocos o malas interpretaciones, motivo por el cual se ha optado por no traducirlo, utilizando el vocablo inglés a lo largo de esta tesis, como se ha podido comprobar en la definición de reutilización de [Peterson, 1991].

de los ingenieros del software, más familiarizados con el vocablo **componente** con el significado de un *elemento software que puede ser directamente incorporado en un número indefinido de aplicaciones.* Sin embargo, el término **componente** está tan sobrecargado que en el área de investigación dedicada a la reutilización sistemática se ha reemplazado por **asset** [Simos et al., 1996]. Así pues, un asset puede definirse como sigue.

1. *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], [Katz et al., 1994].*

2. *Unidad de información con un valor actual o futuro para una empresa de desarrollo o de mantenimiento de software. Los assets pueden incluir una amplia variedad de elementos, tales como productos del ciclo de vida del software, modelos de dominio, procesos, documentos, caso de estudio, resultados de investigación, presentaciones... [DoD, 1995].*

3. *Descripción de una solución parcial (como un componente o un documento de diseño) o conocimiento (como puede ser una base de datos de requisitos o procedimientos de prueba) que los ingenieros utilizan para construir o modificar productos software [Withey, 1996].*

1.1.3 Características de los Assets

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

- **Expresivos:** Los assets están clasificados en un nivel de abstracción adecuado y expresan una utilidad general que los hace susceptibles de ser reutilizados en diferentes contextos o 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 implica que 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 sufrir efectos laterales adversos. Esta propiedad lleva a manejar elementos reutilizables de diferente granularidad; así se tienen los *elementos reutilizables de grano fino*, que son atómicos, es decir no están compuestos por otros, y que serán referenciados a lo largo de este trabajo como *assets*, y los *elementos reutilizables de grano grueso*, que serán aquellos que están formados por un conjunto de assets.

- **Formal:** Los assets debieran poder ser descritos a algún nivel de abstracción mediante una notación formal o semi-formal, de manera que pudiera existir algún mecanismo para poder verificar su corrección, predecir la violación de integridad de sus restricciones a la hora de integrarlo con otros assets o asegurar el nivel de compleción de un producto software construido con assets.

- **Informáticamente representables:** Aquellos elementos software reutilizables que pueden ser descritos en términos de unos valores de unos determinados atributos computacionales, que pueden ser fácilmente descompuestos en partes representables por un ordenador, que pueden ser accedidos, analizados, manipulados y posiblemente modificados por procesos basados en ordenador, tienen un claro potencial para formar parte de una biblioteca flexible de elementos reutilizables. Estos assets pueden ser fácilmente buscados, recuperados, interpretados, modificados y finalmente integrados en sistemas software de mayor entidad.

- **Autocontenidos:** Los assets que encierran una única idea son más fáciles de entender, tienen menos dependencias con factores externos (*ya sean de entorno o de implementación*), tienen unas interfaces fáciles de utilizar, son fáciles de extender, adaptar y mantener.

- **Independientes del lenguaje:** Los elementos software reutilizables de alto nivel de abstracción deben omitir los detalles de implementación propios de los lenguajes de programación. Esto es, los elementos software reutilizables debieran ser descritos en términos de formalismos de especificación, o de forma que las soluciones de bajo nivel pudieran ser utilizadas por una amplia variedad de lenguajes de programación sobre una plataforma de implementación dada.

- **Capaces de representar datos y comportamiento:** Deben ser capaces de encapsular sus estructuras de datos y su lógica hasta un grano fino de detalle, 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.

1.1.4 Bibliotecas de Reutilización y Repositorios

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. De las diversas definiciones de repositorio que se encuentran en la bibliografía se citan las siguientes:

1. *Se define repositorio como una base de datos de información compartida sobre los elementos que se producen o se usan en un desarrollo software [Bernstein and Dayal, 1994].*
2. *Un repositorio es una herramienta para la definición, almacenamiento, acceso y gestión de la información que describe a una empresa y a sus sistemas software, durante cada una de las diferentes fases del ciclo de vida del software [McClure, 1997].*

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 Inc., 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, donde las bibliotecas de reutilización son modelos más aplicaciones o servicios [Wallnau, 1992].

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, 1992a]*".

En la bibliografía 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].

1.1.5 Composición y Generación

Dentro de la reutilización del software existen dos enfoques bien diferenciados: las tecnologías basadas en composición de elementos reutilizables y las tecnologías de generación [Biggerstaff and Richter, 1987]. Tradicionalmente, estas dos líneas de trabajo han presentado caminos separados que sólo en ocasiones puntuales han confluido en soluciones abiertamente híbridas.

Composición

La reutilización basada en composición implica la utilización de componentes, bibliotecas de clases o de funciones... para la construcción de nuevo software mediante la composición *manual* de estos bloques.

Tras el concepto de composición subyace la idea de utilización de elementos software como cajas negras[3], concepto que promueve la idea de ocultación de la información como medio de facilitar la reutilización de módulos software [Parnas, 1972], [Parnas et al., 1989].

La composición es una aproximación de reutilización factible que se refleja en la totalidad de repositorios y bibliotecas de reutilización existentes. No obstante, la composición no

[3]La **reutilización de caja negra** hace referencia a la reutilización tal cual del elemento software, esto es sin entrar a considerar sus contenidos, y por tanto sin modificar el elemento propiamente dicho. La adaptación consiste en el establecimiento de los parámetros del elemento software para satisfacer las necesidades de la aplicación a desarrollar. En este sentido se puede hablar también de **reutilización de caja gris** - cuando para reutilizar un asset se deben aplicar un conjunto de cambios menores a éste - y de **reutilización de caja blanca** - cuando los cambios necesarios para su reutilización son importantes, sufriendo el asset cambios drásticos en estructura y/o comportamiento [Karlsson, 1995].

consigue en general altos niveles de reutilización al encontrarse con los límites e inhibidores impuestos por los elementos software que se manejan. En este sentido, la tecnología de objetos ofrece excelentes mecanismos que a priori pueden potenciar la reutilización basada en composición.

La aproximación más práctica dentro del apartado de la composición se ve reflejada en la utilización de elementos software ya creados, con independencia de su nivel de abstracción, en nuevos esfuerzos de desarrollo; surgiendo una nueva vertiente de desarrollo dedicada al desarrollo de assets o a su identificación mediante técnicas de reingeniería y de *minería de assets*. Experiencias en este campo se tienen de la mano de las numerosas bibliotecas de reutilización existentes, tales como ASSET, PAL, CARDS, COSMICS, DSRD o STARS, así como en los marcos teórico/prácticos establecidos por importantes proyectos de investigación, entre los que cabe citar a:

- **EEC-SPRIT II ITHACA** (*Integrated Toolkit for Highly Advanced Computers Applications*) [Ader et al., 1990]. Proyecto europeo que 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 objetos y la reutilización.

- **REBOOT** (*REuse Based on Object-Oriented Techniques*) [SER Consortium, 1996], [Karlsson, 1995]. 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 instituciones en las que se desarrolla software.

- **EUROWARE** (*Enabling Users to Reuse Over Wide AREas*) [Sema Group, 1996], [SER Consortium, 1996]. 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.

- **RBSE** (*Repository Based Software Engineering*) [Eichmann, 1995]. 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.

- **STARS** (*Software Technology for Adaptable, Reliable Software*) El programa de reutilización STARS es un proyecto que desde hace años viene siendo soportado por el

Departamento de Defensa de EEUU. Se centra en la integración de tres áreas: el proceso, la arquitectura y la reutilización. Utiliza como nexo común un ciclo de vida del software basado en líneas de producto, soportado por cuatro pilares:

1. *Conducido por proceso:* La producción de software de calidad debe producirse bajo unas restricciones de gestión incluidas dentro de un proceso sistemático.
2. *Centrado en la arquitectura:* La reutilización comienza en el nivel arquitectónico, fijando que assets son potencialmente reutilizables.
3. *Específico de un dominio:* Se alcanzan mayores cuotas de reutilización dentro de un dominio, no entre dominios.
4. *Basado en bibliotecas de reutilización:* Los assets se almacenan y se organizan en bibliotecas de reutilización (*repositorios*), que permite un acceso fácil a los desarrolladores con reutilización.

En el aspecto tecnológico STARS incluye un marco conceptual para los procesos de reutilización[4] **CFRP** (*Conceptual Framework for Reuse Processes*) [Creps et al., 1992], [STARS, 1993], un método de análisis de dominio **ODM** (*Organization Domain Modeling*) [Simos et al., 1996], un método de análisis de dominio basado en características **FODA** (*Feature Oriented Domain Analysis*) [Kang et al., 1990], diferentes guías y manuales sobre reutilización dentro de **CARDS** (*Comprehensive Approach to Reusable Defense Software*) [Wallnau, 1992], [Petracca and Bock, 1994], [Gregory, 1995], una arquitectura de software específica del dominio **DSSA** (*Domain-Specific Software Architecture*) [Tracz et al., 1993] y un modelo para interoperabilidad entre bibliotecas de reutilización ALOAF (*Asset Library Open Architecture Framework*) [Solderitsch et al., 1992a].

Una aproximación mucho más formal a la tecnología de composición viene de la mano de los **Esquemas Software** [Krueger, 1992]. En los esquemas software el énfasis reside en la reutilización de estructuras de datos y algoritmos abstractos más que en la reutilización de código. El Sistema PARIS [Katz et al., 1989] es un buen representante de esta tecnología. El esquema de reutilización en PARIS se instancia para producir el código fuente, de forma que cada esquema consta de una descripción semántica formal del esquema, aserciones para la correcta instanciación del esquema y aserciones para la validación del esquema instanciado.

[4] **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 CFRP se puede ofrecer una base para la adopción y mejora de las prácticas de reutilización dentro de una organización.

Generación

La aproximación de la reutilización por generación asume que es posible la descripción de una arquitectura genérica (*abstracta*) para el software que posteriormente puede ser, automáticamente o semiautomáticamente, compuesto para producir software de trabajo (*concreto*) mediante herramientas de generación [Sant'Anna et al., 1998].

En los sistemas basados en generación es mucho más difícil identificar cual es el componente concreto que se está reutilizando, porque lo que realmente se está reutilizando es la *estructura* y el *conocimiento* inmerso en un software que *genera* otras aplicaciones software.

Dentro de los sistemas basados en generación se distinguen tres tipos, dependiendo de la propiedad que éstos enfaticen [Biggerstaff and Perlis, 1989], [Krueger, 1992]:

- **Sistemas basados en lenguajes de muy alto nivel (*VHLL*):** Se utilizan lenguajes de especificación definidos formalmente, en los que representar dominios de problemas, ocultando los detalles de implementación, elevando el nivel de discusión al dominio del problema. Estos lenguajes se basan en abstracciones matemáticas que no se suelen emplear en el ciclo de vida tradicional del software.

  Las construcciones de estos lenguajes sirven como especificaciones abstractas de elementos reutilizables, mientras que la salida del generador se corresponde con abstracciones del nivel de implementación.

  Un ejemplo tradicional de lenguaje VHLL es SETL, basado en teoría de conjuntos [Kruchten et al., 1984], [Dubinsky et al., 1989].

  Sin embargo, una referencia más actual y cercana al contexto en el que se desarrolla esta tesis es el trabajo realizado en la **UPV** (*Universitat Politècnica de Valencia*) en el campo de la programación automática [Ramos et al., 1995], más concretamente en el apartado de la generación de código a partir de especificaciones realizadas en el lenguaje de especificación orientado a objetos **OASIS** (*Open and Active Specification of Information Systems*) [Pastor and Ramos, 1995], [Letelier et al., 1998], [Sánchez et al., 1998].

  OASIS aporta un enfoque formal para la especificación de modelos conceptuales siguiendo el paradigma orientado a objetos. De forma global, se puede decir que OASIS se basa en la Lógica Deóntica (*la lógica de las obligaciones, prohibiciones y permisos*). Las especificaciones OASIS son, esencialmente, un conjunto estructurado de definiciones de clase, donde la plantilla completa de la clase se corresponde con fórmulas en una variante de la Lógica Dinámica.

- **Generadores de aplicaciones:** Encierran en su diseño un patrón arquitectónico que será reutilizado al generar instancias específicas de sistemas finales; de forma que todas las instancias tendrán dicho patrón en común. La entrada de estos generadores de aplicaciones son abstracciones de muy alto nivel perteneciente a un dominio muy concreto. Se diferencian de los VHLL en que éstos son independientes de la aplicación, mientras que los generadores de aplicaciones se encuentran muy ligados a un dominio.

  Como representante de los generadores de aplicaciones se puede tomar uno de los trabajos más clásicos dentro de la reutilización basada en generación, **la máquina DRACO** propuesta por *James M. Neighbors* [Neighbors, 1984], [Neighbors, 1991].

  La máquina DRACO podría haberse puesto como ejemplo de cualquiera de los tres tipos de sistemas basados en generación porque recibe como entrada una especificación escrita en un lenguaje dependiente de un dominio, con el que el usuario expresa su problema; genera programas a partir de dichas especificaciones; y para llevar a cabo la generación se basa en un conjunto de transformaciones.

- **Sistemas de transformación:** Estos sistemas se centran en el papel, la estructura y las operaciones propias de las transformaciones necesarias para pasar de las especificaciones de alto nivel a las especificaciones de implementación.

  Según [Partsch and Steinbruggen, 1983], una transformación es una operación de mapeo de programas en programas. La aplicación de una transformación T a un programa P puede expresarse como:

  $$P \times T \rightarrow P'$$

  Donde P' es el resultado de la aplicación de la transformación y es semánticamente equivalente a P, pero más eficiente [Cheatham, 1984].

  Actualmente, uno de los trabajos más interesantes en el área de los sistemas de transformaciones es el realizado en la Universidad Pontificia de Rio de Janeiro por el Dr. do Prado Leite y su grupo de investigación, que han retomado la idea de la **Máquina DRACO** [Neighbors, 1984] para establecer transformaciones entre diferentes dominios software a través de lo que han denominado **Máquina Draco-PUC** [Do Prado Leite et al., 1994], y que está siendo actualizada para ser operativa en el mundo del desarrollo orientado a componentes (*componentware*) con su acercamiento a CORBA [Sant'Anna et al., 1998].

Reutilización Híbrida

La reutilización híbrida combina tanto la aproximación por composición como por generación, completando el conjunto de elementos reutilizables con lenguajes específicos del dominio y con generadores de aplicaciones.

Con este enfoque se pueden ensamblar aplicaciones seleccionando elementos reutilizables dentro de una arquitectura de domino (*tales como plantillas o marcos*), o generando elementos reutilizables para algunas partes de la aplicación mientras que se utiliza explícitamente composición manual para otras [Batory et al., 1994].

Dentro de la reutilización híbrida es importante hablar de los **kits híbridos específicos de un dominio** propuestos por Martin Griss y Kevin Wentzel [Griss and Wentzel, 1993], [Griss and Wentzel, 1994], [Griss and Wentzel, 1995].

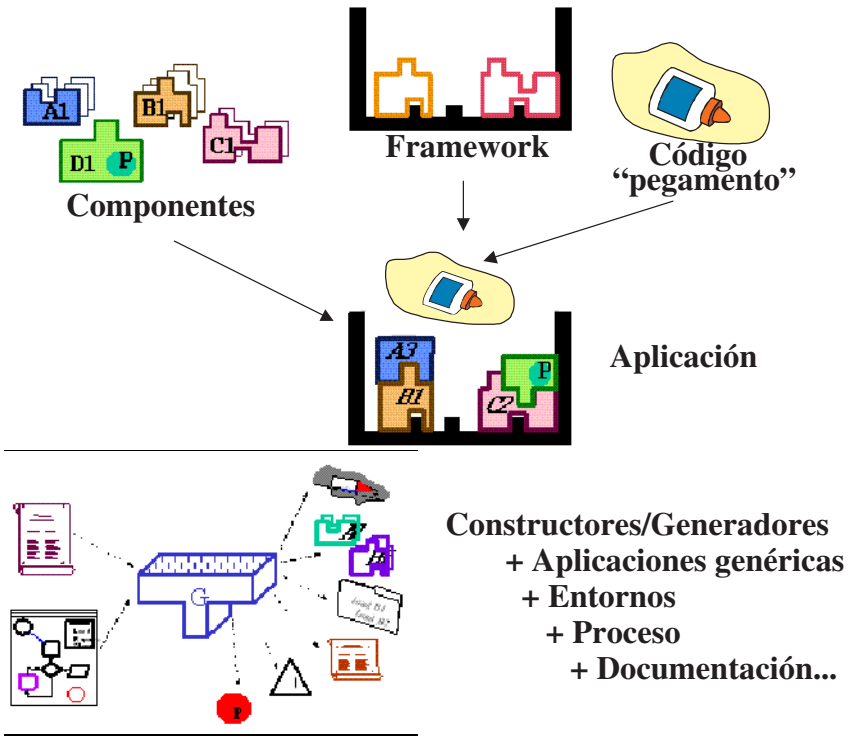

Figura 1.1: Esquema básico de un kit híbrido de dominio [Griss and Wentzel, 1994].

El concepto de **kit** significa la reunión completa y coherente en un *paquete* de diferentes *productos* reutilizables que se ajusten bien unos con otros, simplificando la construcción de aplicaciones software, como se representa en la figura 1.1. La noción de **híbrido** indica que dentro del *paquete* se combinan elementos generadores y constructores, representantes

de las dos principales corrientes de reutilización. Por último, la acepción de **específico de un dominio** establece que los productos que forman el *kit* están diseñados para construir aplicaciones dentro de una misma familia o dominio.

Otra forma de reutilización híbrida se tiene en los **generadores basados en composición** [Biggerstaff, 1999], donde se tiene un generador que es capaz de generar elementos reutilizables mediante la composición de bloques primitivos.

1.1.6 Ciclo de Vida de la Reutilización

Los modelos de ciclo de vida del software tradicionales se han concebido bajo la premisa de que se iban a aplicar en el desarrollo de un producto software concreto, comenzando el desarrollo desde cero. Estos modelos generalmente se estructuran en una serie de fases que van desde los estudios de viabilidad hasta la implantación y mantenimiento del producto.

Los procesos que hacen uso de la reutilización dentro de un dominio de aplicación presentan una perspectiva diferente. Se busca partir de un conjunto de elementos ya existentes y que pertenecen al mismo dominio de aplicación que el producto software que se quiere crear. El cambio de un único producto a varios productos que comparten unas características formando una familia, tiene una gran repercusión en el proceso de desarrollo. La mayor consecuencia es que el proceso de desarrollo que soporta reutilización tiende a estructurarse en dos procesos distintos y separados: *Ingeniería de Dominio* (*Domain Engineering*) e *Ingeniería de Aplicación* (*Application Engineering*).

El motivo principal para contar con dos procesos separados es que cada uno de ellos tiene unos objetivos diferentes. Mientras que la Ingeniería de Dominio se centra en el desarrollo de los elementos reutilizables que forman una familia de productos, la Ingeniería de Aplicación se dirige hacia la construcción o desarrollo de un producto software individual, dentro de la familia de productos definida, y que satisface un conjunto de requisitos y restricciones de un usuario específico, adaptando e integrando los elementos reutilizables existentes.

Estos procesos de desarrollo de aplicaciones software serán más efectivos en aquellas organizaciones donde se tenga un dominio de aplicación (*también denominado dominio de negocio*) bien definido, con subdominios (*a los que normalmente se suele referir como líneas de producto*) perfectamente identificados.

Existen diferentes aproximaciones a la reutilización que ponen mayor énfasis en unas facetas que en otras y donde la terminología empleada difiere. Es muy común encontrarse

en la bibliografía especializada con dos subprocesos, equivalentes en esencia a la Ingeniería de Dominio y a la Ingeniería de Aplicación, el primero de ellos se encarga del **desarrollo de software reutilizable**, mientras que el segundo de **la reutilización del software existente**. La figura 1.2 recoge un esquema general de este proceso básico de reutilización, adaptado del propuesto en [Karlsson, 1995].

El primer subproceso recibe el nombre de *Desarrollo **Para** Reutilización*, y sería equivalente a la *Ingeniería de Dominio*, ocupándose de la construcción de assets para ser utilizados en el desarrollo de aplicaciones similares dentro de un dominio concreto; mientras que el segundo subproceso, se denomina *Desarrollo **Con** Reutilización*, y equivaldría a la *Ingeniería de Aplicación* [Girardi, 1992], [Karlsson, 1995], [McClure, 1997].

Figura 1.2: Ciclo de vida de la reutilización.

El desarrollo para reutilización es una tarea interdisciplinaria. Requiere de un buen conocimiento del dominio seleccionado, así como experiencia en el desarrollo de aplicaciones en el mismo dominio.

El desarrollo para reutilización y con reutilización constituyen actividades complementarias pero relacionadas. De hecho, el desarrollo para reutilización puede verse como una

labor de evolución que utiliza la experiencia adquirida y acumulada en el desarrollo con reutilización; experiencia que sirve como realimentación y cierre del ciclo de reutilización.

Las principales actividades que involucradas en el desarrollo para reutilización son la *cualificación*, la *generalización*, la *clasificación* y la *evolución*, como muestra la figura 1.3, mientras que las propias del desarrollo con reutilización son la *recuperación*, la *adaptación* y la *integración* o *composición*, como se aprecia en la figura 1.4, [Girardi, 1998].

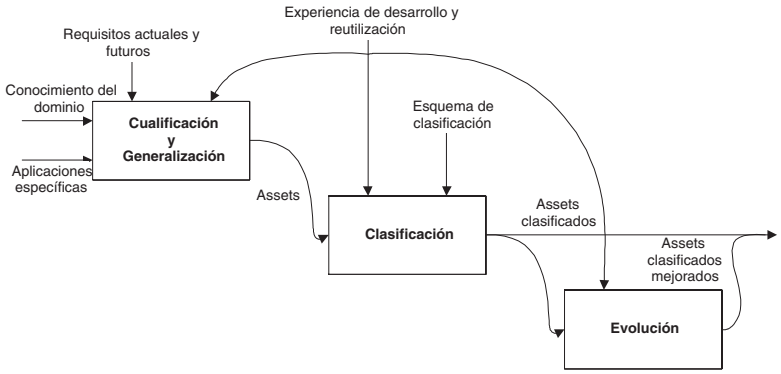

Figura 1.3: Actividades en el desarrollo para reutilización [Girardi, 1998].

- **Cualificación y Generalización.** Los repositorios se pueblan con assets los cuales:
  - Ya existen y se identifican como 'reutilizables' de acuerdo a un conjunto definido de métricas que determinen su reusabilidad, o
  - Se construyen elementos software reutilizables de propósito general identificando características comunes de elementos software concretos.

  En relación con la generalización cabe destacar la importancia que tiene para la reutilización, especialmente en el diseño de jerarquías de clases en la orientación a objetos, así una metodología orientada a objetos como **MOSES** [Henderson-Sellers and Edwards, 1994] la tiene presente en sus actividades.

  Sin embargo, la generalización suele tener un sentido de actividad a realizar después de que el proyecto software ha terminado, para aprovecharse de los resultados en futuros desarrollos, con lo cual incluye un factor de coste añadido que en la mayoría de las veces acaba con su aplicación. Para evitar esto se podría pensar en la generalización como una actividad que se lleva a cabo a lo largo de todo el ciclo de vida del proyecto, pudiéndose utilizar los resultados en el mismo proyecto y sin un coste final añadido [Henderson-Sellers and Pant, 1993].

- **Clasificación.** Proceso por el cual los assets de un repositorio se catalogan en clases disjuntas siguiendo un criterio de emparejamiento de descriptores. Todos los miembros de cada grupo producido por la clasificación comparten al menos una característica que no está compartida por los miembros de los otros grupos. La herramienta que determina como realizar la clasificación en una configuración dada, los conjuntos de descriptores y la posible ordenación interna, emparejando criterios y reglas para la asignación de clases, se denomina **esquema de clasificación**.

- **Evolución.** Los contenidos de un repositorio deben evolucionar para adaptarse a la evolución de los dominios de aplicación que cubren, así como para mejorar sus contenidos. Esto requiere:

  – Mantener los assets susceptibles de ser reutilizados y eliminar los obsoletos;

  – Reclasificar o reorganizar los assets para mejorar su recuperación;

  – Mejorar la información disponible para entender y adaptar los assets; y

  – Rediseñar los assets para hacerlos más reusables.

  Para lograr estos se necesita una realimentación del repositorio con la experiencia práctica del proceso de desarrollo con reutilización.

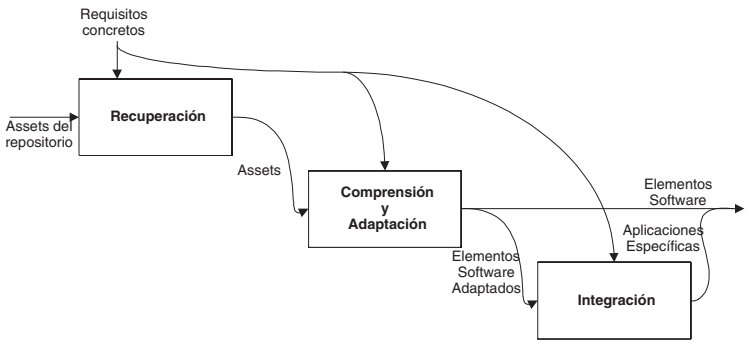

Figura 1.4: Actividades en el desarrollo con reutilización [Girardi, 1998].

- **Recuperación.** Mediante este proceso los potenciales elementos reutilizables que satisfacen unos requisitos específicos del desarrollador con reutilización, son buscados y seleccionados del repositorio.

- **Comprensión y Adaptación.** Un asset recuperado puede reutilizarse tal cual como una caja negra o bien especializado para unos requisitos particulares. La tarea

de adaptación está directamente relacionada con la comprensión del asset; cuanto más difícil sea entender un asset, más complicada será su adaptación a unos nuevos requisitos.

- **Integración.** Los assets seleccionados son ensamblados para construir elementos software más complejos, tales como subsistemas software o aplicaciones específicas.

Para J. L. Cybulski [Cybulski, 1996] el proceso de reutilización del software está formado por tres fases del proceso de construcción de assets, esto es, *análisis, organización* y *síntesis*. Las tareas asociadas a estas tres fases se pueden apreciar en la figura 1.5, coincidiendo con las actividades vistas desde la perspectiva del desarrollo para reutilización y con reutilización.

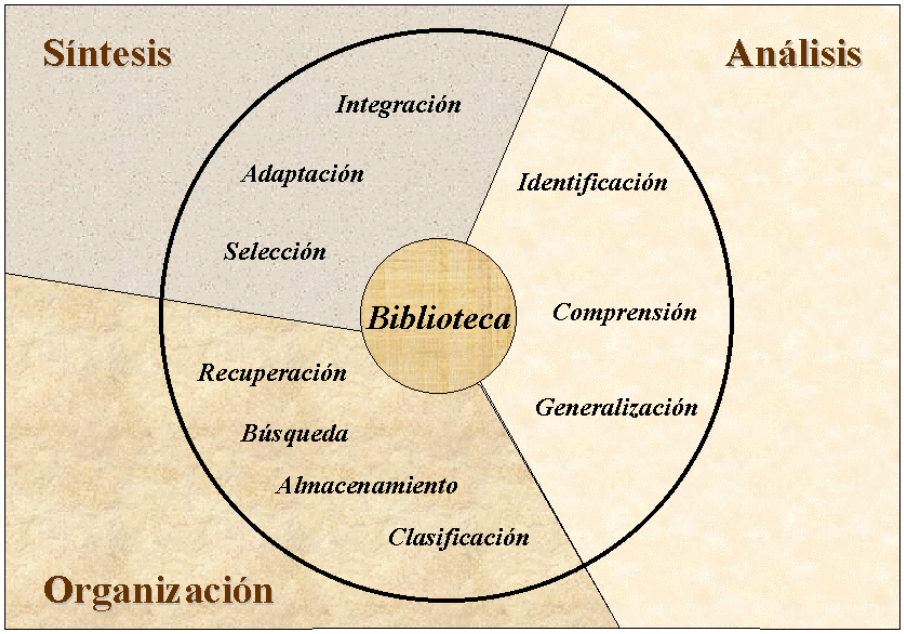

Figura 1.5: Proceso de reutilización del software [Cybulski, 1996].

En la fase de análisis se comienza con la identificación de los assets en productos software existentes o en un dominio ya estudiado, a lo que sigue la comprensión y representación de los mismos en algún formalismo adecuado para reflejar su función y semántica, con una posible generalización que amplíe el ámbito de su futura reutilización. La organización de assets es la fase que incluye la clasificación y el almacenamiento de los assets en un repositorio, así como la subsiguiente búsqueda y recuperación de assets cuando sea necesario. Por último, la fase de síntesis de assets consiste en la selección de los assets candidatos

recuperados, su adaptación para ajustarse a las nuevas aplicaciones y su integración dentro de un producto software completo.

Reutilización en los Métodos de Desarrollo

Para contar con una verdadera reutilización sistemática dentro de la tecnología de objetos, es necesario contar con unos métodos de desarrollo que incluyan de forma natural y efectiva la reutilización dentro de sus procesos [Griss, 1995c], [Griss, 1995d].

La mayor parte de los métodos de desarrollo existentes, ya sea en el paradigma estructurado o en el objetual, se pueden considerar *indiferentes* a la reutilización.

Dado que no existe un modelo de proceso ideal para todas las situaciones, se puede partir de diferentes ciclos de vida (*macroprocesos*) que sirvan como marcos de referencia para incluir elementos concretos (*microprocesos*) que describan de forma específica roles, actividades, métodos, métricas, responsabilidades y elementos a obtener de los gestores y desarrolladores involucrados. Estos microprocesos deben contemplar tanto la parte de Desarrollo Para Reutilización, con especial hincapié en las técnicas de Ingeniería de Dominio, como la parte de Desarrollo Con Reutilización.

Uno de los primeros intentos de obtener una metodología para reutilización se tiene en [Kang et al., 1992] donde se propone una metodología basada en el modelo de ciclo de vida MIL-STD-2167A [DoD, 1988]. Otra experiencia interesante es el proyecto ESPRIT REBOOT, que tiene como objetivo crear un marco metodológico y organizador para integrar la reutilización como un método habitual en el desarrollo de software, principalmente desde la perspectiva de la orientación a objetos [Karlsson, 1995].

Dentro de los métodos clásicos de Análisis y Diseño Orientados a Objetos por su relación con la reutilización se ha de destacar **OOSE**[5] (*Object Oriented Software Engineering*) [Jacobson et al., 1992], basada en casos de uso, con facilidades para la trazabilidad y extracción y empaquetado de subsistemas. Posteriormente, OOSE se extiende para dar soporte a la reutilización sistemática a gran escala con **RSEB** (*Reuse-Driven Software Engineering Business*) [Griss, 1996a], [Jacobson et al., 1997]. RSEB aporta a OOSE:

- Una visión arquitectónica para el establecimiento de líneas de productos partiendo de modelos OOSE ya construidos y validados.
- Procesos de reutilización iterativos e incrementales para la construcción, uso, gestión y mantenimiento de assets.

[5] También conocida como **Objectory**.

- Un modelo de organización para el trabajo de varios equipos de personas, reutilizando assets realizados por otros equipos.

Tanto la parte de proceso como de organización ha sido modelada utilizando **OO BPR** (*Object-Oriented Business Process Reengineering*) [Jacobson et al., 1994].

La creación de RSEB ha sido paralela al nacimiento de UML, teniendo a Ivar Jacobson como nexo de unión entre los dos eventos. Esto se traduce en la utilización de UML como lenguaje de modelado en RSEB, pero también se ven influencias de RSEB en el método unificado propuesto por Rational, **RUP** (*Rational Unified Process*) [Jacobson et al., 1999].

Sin embargo, uno de los puntos débiles de RSEB está en su carencia de modelos adecuados para la Ingeniería de Dominio, para lo cual se pueden utilizar métodos específicos de Ingeniería de Dominios como pueden ser FODA [Kang et al., 1990], o el método de Ingeniería de Dominio de segunda generación ODM 2.0 [Simos et al., 1996], [Simos and Anthony, 1998]. Precisamente, para completar RSEB con características de Ingeniería de Dominio se está integrando FODA en los procesos y elementos de trabajo de RSEB [Griss et al., 1997], [Griss et al., 1998].

Para terminar con este repaso por los métodos de desarrollo orientados a objetos más involucrados con la reutilización, no podía dejar de mencionarse a **Catalysis** [D'Souza and Wills, 1999], que propone un método centrado en el desarrollo basado en componentes ***CBD*** - *Component Based Development*, haciendo un uso intensivo de técnicas de diseño orientado a objetos, especialmente de *frameworks* y *patrones*.

1.1.7 Elementos Reutilizables en las Fases del Ciclo de Vida del Software

Los elementos software reutilizables no tienen que limitarse al nivel de abstracción de implementación. De hecho el aumento de la potencia de la reutilización pasa por elevar el nivel de abstracción de los elementos reutilizables, así David Parnas [Parnas et al., 1989] afirma que la construcción y catalogación de abstracciones incrementa la capacidad de reutilización del software, dado que los desarrollos de una abstracción pueden ser reutilizados para cualquier modelo válido de la abstracción.

A continuación se hace un recorrido por los principales elementos reutilizables propios de las diferentes fases del ciclo de vida del software, presentándolos de mayor a menor nivel de abstracción.

Análisis de Dominio

Los elementos reutilizables de mayor nivel de abstracción son los que se obtienen mediante el análisis del dominio de un problema. El análisis de dominio surge de la idea de que en un caso real la reusabilidad no es una propiedad universal de un asset, sino que depende del contexto del problema y de su solución, que son relativamente cohesivos y estables [Arango and Prieto-Díaz, 1991].

El objetivo principal del análisis de dominios es la construcción de un modelo de dominio cuyos componentes puedan ser reutilizados en la resolución de una amplia gama de problemas. Este modelo en términos generales debe incluir la definición de los conceptos utilizados en la especificación de los problemas y de los sistemas software dentro del dominio, la definición de decisiones típicas de diseño, alternativas, decisiones y sus justificaciones y planes de implementación.

A partir de mediados de la década de las noventa se ha comenzado a utilizar con una alta frecuencia un término más genérico que análisis de dominio, la *Ingeniería de Dominio*; siendo su principal objetivo la optimización del proceso de desarrollo de software sobre una "*ventana*" de múltiples aplicaciones dentro de un área de negocio común o un dominio de problema [Simos et al., 1996].

Existen diferentes trabajos en la bibliografía donde se recogen comparaciones sobre los métodos y tecnologías de la ingeniería del dominio [Prieto-Díaz and Arango, 1991], [Wartik and Prieto-Díaz, 1992]. A los trabajos contemplados en estos estudios debe añadirse ODM 2.0 (*Organization Domain Modeling*) [Simos et al., 1996].

Sin embargo, no existe un consenso en las definiciones de los conceptos propios de la Ingeniería de Dominio, de forma que términos como análisis de dominio, ingeniería de dominio y modelado de dominio se utilizan de forma inconsistente en la bibliografía. Normalmente, el término Ingeniería de Dominio hace referencia a todo el conjunto de procesos destinados a la creación de assets específicos del dominio (*lo que en otros ámbitos se denomina desarrollo para reutilización*). De los procesos propios de la Ingeniería de Dominio, el primero es el *análisis de dominio*, seguido de los subsiguientes procesos (*arquitectura de dominio e implementación de dominio*). No obstante, para otros autores el término Ingeniería de Dominio hace referencia a la implementación de dominio exclusivamente y, para introducir más confusión, hay quien defiende que el término análisis de dominio se refiere a la adquisición de datos del dominio y el modelado del dominio a la representación de estos datos.

Para no caer en esta caótica maraña terminológica, en esta propuesta de tesis doctoral

se va a seguir la terminología según ODM [Simos et al., 1996], que utiliza sólo unos pocos términos con un significado concreto. Así, todo el proceso va a denominarse Ingeniería de Dominio, el cual consiste en tres fases principales: *planificación de dominio, modelado de dominio* e *ingeniería de asset base*. El término modelado de dominio recoge tanto la adquisición como la representación del conocimiento del dominio[6].

ODM presenta un concepto de dominio inspirado en una aproximación etnográfica, donde la cultura estudiada es la de los ingenieros del software que construyen las aplicaciones software y los usuarios que las utilizan en sus lugares de trabajo. Con esta aproximación modela el dominio en el que los ingenieros del software están interesados, incluyendo el estudio de la "*escena cultural de los usuarios finales del sistema*" (*el dominio del mundo real*), los "*assets culturales*" (*requisitos, diseños, implementaciones y otros documentos*) y, si es posible, la "*escena cultural de los desarrolladores*" (*el entorno en que los requisitos son analizados, las aplicaciones son creadas y mantenidas...*). La Ingeniería de Dominio tiene lugar en la mayoría de los entornos de negocio donde las organizaciones tienen objetivos estratégicos. Así, ODM utiliza modelos explícitos del contexto social y organizativo para dar soporte a un modelo completo de ciclo de vida en la ingeniería de dominio.

Diferentes aproximaciones defienden que un determinado dominio conlleva un conjunto de aplicaciones completas, lo que se denomina aproximación de familias de sistemas. ODM explícitamente anima a la identificación y selección de dominios en un nivel de subsistema, o lo que es lo mismo el soporte de dominios de grano más fino. Un dominio definido al nivel de organización es demasiado grande y requiere un nivel insostenible de compromiso por parte de la organización.

Otro beneficio de esta aproximación es que, si el modelado del dominio se realiza como propulsor de la creación de assets, el nivel de subsistema frecuentemente será el mayor candidato para la realización de una reingeniería.

El concepto de subdominio de ODM puede considerarse como muy cercano al concepto de línea de producto. Las líneas de producto son un área dentro de la reutilización que está actualmente en expansión. Los conceptos de dominio y línea de producto están sumamente relacionados, pudiéndose decir que ambos referencian al mismo nivel de abstracción, determinándose las fronteras de cada término en función de los roles que desempeñen las personas que los utilizan [Poulin, 1997].

Una línea de producto se define como sigue:

[6] El término análisis de dominio se ha excluido por su imprecisión debida a su sobrecarga de significados.

1. *Grupo o familia de productos relacionados realizados por el mismo proceso y para el mismo propósito, diferenciándose sólo en el estilo, modelo o tamaño. Una línea de producto agrupa aplicaciones relacionadas en familias de aplicaciones tomando ventaja de los elementos comunes de la familia. Dado que los productos están relacionados (tienen funcionalidad o requisitos de usuario similares) hay un alto grado de aspectos comunes en una línea de producto [Sonnemann, 1995].*

2. *Colección de productos software que recogen un conjunto de requisitos de sistema comunes, y están organizadas alrededor de una actividad específica de negocio [Cohen et al., 1995].*

3. *Grupo de productos que comparten un conjunto de características comunes gestionadas, que satisfacen las necesidades específicas de un sector concreto del mercado [Bass et al., 1997].*

La forma más económica de construir una línea de producto es como una *familia de productos*, donde una familia de productos es un grupo de sistemas construidos a partir de una conjunto común de assets [Bass et al., 1997], [Bass et al., 1998], [Bass et al., 1999].

Otras referencias que resumen el trabajo que se está realizando en relación con las líneas de trabajo son [Clements, 1997a] y [Bergey et al., 1998].

Análisis de Requisitos

El nivel de abstracción de análisis de dominio se centra en el estudio del dominio de aplicación y de las líneas de producto que se pueden establecer en dicho dominio. Por contra, el nivel de análisis se centra en los assets relacionados con la documentación y especificación de los requisitos de un producto software.

La reutilización de elementos de este nivel de abstracción promete importantes ventajas, entre las que se destaca la trazabilidad hacia assets clasificados en niveles de menor abstracción [Matsumoto, 1989].

La reutilización de assets procedentes de las primeras fases del ciclo de vida supone un gran beneficio en el desarrollo de proyectos software. Sobre estos elementos software recae un alto porcentaje del esfuerzo de todo el proyecto software, lo que implica que su reutilización debe aportar un gran ahorro económico y un aumento de la productividad [Cybulski, 1997], [Cybulski et al., 1997b].

Existen diferentes factores que influyen en la eficiencia de las fases de elicitación y

especificación de los requisitos, entre los que cabe citar [Christel and Kang, 1992], [SEI, 1991], [Gotel and Finkelstein, 1994]:

- *La comunicación entre todos los implicados:* Entran en juego personas con roles muy diferenciados entre los que se establece una comunicación en lenguaje natural con vocabularios muy dispares.
- *La trazabilidad:* Que se ve dificultada por la complejidad y frecuencia de las revisiones del producto software, la pérdida de la forma inicial de los requisitos...
- *Los procesos iniciales de los desarrollos:* Complicados de controlar por la falta de rigor, definición y formalización de los procesos de negocio, así como de los procedimientos de las organizaciones, y el pobre conocimiento de las relaciones entre las personas involucradas en el proceso y sus responsabilidades y cometidos.

Estos problemas tienen su origen en la diversa naturaleza y forma de los documentos de especificación de requisitos, lo cual se traduce en los diferentes medios utilizados para su recolección, la falta de estructura y el uso de notaciones inapropiadas.

- *Medios:* Los documentos iniciales utilizan formatos que facilitan la comunicación entre los implicados. Incluyen textos en lenguaje natural, gráficos, imágenes, vídeos, animaciones, diálogos, sonidos... En general estos elementos no suelen estar en formatos que directamente se puedan incorporar a un sistema basado en computadores, siendo su automatización, formalización e integración una tarea difícil.
- *Estructura:* Los documentos producidos en las fases iniciales del desarrollo se caracterizan por una completa falta de estructura formalizada o de organización lógica.
- *Notación:* El medio tradicional de almacenamiento de los requisitos es el lenguaje natural porque es el medio natural de comunicación entre los implicados. La utilización del lenguaje natural, aunque necesaria, introduce "*ruido*" debido a redundancias, "*silencio*" por omisiones, "*sobre-especificaciones*" por la inclusión de detalles de la solución, "*contradicción*" causada por incompatibilidades, "*ambigüedad*" debida a las múltiples interpretaciones que se pueden hacer de un texto en lenguaje natural, "*falsas esperanzas*" causadas por falta de una forma de validación realista y "*referencias adelante*" a conceptos que se introducirán más adelante [Meyer, 1985].

Sin embargo, a pesar de todos los problemas presentados, la reutilización de requisitos y de sus especificaciones conllevan unas importantes contraprestaciones sobre todo el de-

sarrollo, que se reflejan especialmente en el coste y la calidad del producto final, aunque se pueden citar, entre otras, las siguientes ventajas añadidas:

- *Mejora de la calidad de las especificaciones de requisitos.*
- *Mejor utilización de los recursos disponibles.*
- *Introducción de la reutilización sistemática a través de todo el ciclo de vida.*
- *Asistencia de desarrollo en los primeros momentos del ciclo de vida.*
- *Facilidad para la reutilización de los elementos del ciclo de vida derivados de éstos.*

Los assets que tienen cabida en este nivel de abstracción son de muy diverso tipo y naturaleza (*planes de estrategia, evaluación de riesgos, workflows, documentos de requisitos, redes PERT o CPM...*), una clasificación más detallada de éstos se encuentra en [Cybulski et al., 1997a].

Diseño

La fase de diseño es fundamental para la consecución de assets de implementación. Así, los elementos software construidos en esta fase tienen una total validez como assets por los siguientes motivos:

1. *El diseño debe hacerse con miras a la consecución de componentes reutilizables, apareciendo el concepto de diseño para reutilización.*
2. *El código fuente no contiene generalmente demasiada información del diseño original que sirva para construir nuevos elementos software reutilizables a partir de los ya existentes [Biggerstaff, 1989].*

Uno de los tipos de elementos software que tradicionalmente más se han prestado para su reutilización han sido los **TAD** (*Tipo Abstracto de Dato*), que son el soporte natural para estructuras de datos tales como listas, pilas, árboles o conjuntos. Estos TAD, gracias a su función de utilidad, se consideran independientes de un dominio de aplicación concreto, convirtiéndose en el mayor exponente de la reutilización horizontal o reutilización en varios dominios de aplicación diferente.

Los TAD pueden desarrollarse desde dos puntos de vista diferentes: como un conjunto de estructuras de datos más los algoritmos que las manejan, estando especificados de forma

independiente de su implementación, normalmente utilizando algún lenguaje de especificación algebraica; o como bibliotecas de módulos que están disponibles para su inmediata utilización en un lenguaje de programación. El primer enfoque se denomina *teórico* pudiéndose encontrar numerosos libros que representan la forma teórica de los TAD, entre los cuales se pueden citar los ya clásicos [Knuth, 1973a], [Knuth, 1973b] y [Aho et al., 1983]. El segundo enfoque se denomina *práctico*, encontrándose ejemplos en [Kleine, 1986] o en [Booch, 1987].

Ambas formas tienen sus pros y sus contras. Las formas teóricas ofrecen una mayor profundidad para contemplar todas las variantes de implementación, pero en su contra tienen que localizar el TAD adecuado para un determinado problema no siempre es fácil, con el añadido de que la elección se tiene que hacer previamente a la codificación.

Las formas prácticas son más fáciles de seleccionar, aunque no ofrecen un ámbito tan rico de diferentes representaciones e implementaciones, pudiendo existir un coste de la eficiencia de las implementaciones existentes.

En [Uhl and Schmid, 1990] se presentan los problemas de creación y localización de los TAD en relación con su reutilización, además de abordar cómo organizar un catálogo de TAD reutilizables.

El concepto de genericidad va muy unido al de los contenedores que se ven expresados por los TAD más extendidos, de forma que se definen independientemente de su contenido dentro de un entorno de programación concreto. Ejemplos destacados de bibliotecas que hacen uso de este concepto son la **Ada Generic Library** [Musser and Stepanov, 1989] para **Ada**, la **STL** (*Standard Template Library*) [Lee and Stepanov, 1995] para **C++** o la **ObjectSpace JGL**™ (*Java Generic Library*) [ObjectSpace, 1999] para **Java**.

El diseño orientado a objetos tiene como uno de sus objetivos más importantes la potenciación de la reutilización del software. Tomando como base los elementos intrínsecos del modelo objeto, se construyen grupos de clases que interaccionan entre sí para la consecución de un fin común.

De los elementos que componen un modelo objeto, aquellos que potencian la reutilización en el diseño orientado a objetos son:

- **Generalización/Especialización.** La especialización guía al diseñador en la reutilización de una abstracción existente, para diseñar una clase más específica. La generalización, por el contrario, posibilita la creación de superclases, a partir de clases existentes, que describen sus elementos comunes utilizando un proceso de factoriza-

ción. La especialización no implica la redefinición de las clases existentes; aunque el proceso de generalización sí implica la modificación de las definiciones existentes, porque los elementos comunes entre las clases se trasladan a un nivel más alto de definición.

La relación de generalización/especialización se implementa en los lenguajes de programación orientados a objetos mediante el mecanismo de herencia. El uso de la herencia como mecanismo de reutilización se concibe como de *caja blanca* porque se rompe el principio de ocultación de la información de los ancestros, convirtiéndose con su mal uso en una forma poco apropiada de reutilización [D'Souza and Wills, 1999], que puede obtener beneficios a la hora de extender código existente, pero pagando el precio de un diseño mucho más oscuro.

Sin tomar posiciones extremistas, se debe reconocer la importancia del uso de la relación de generalización/especialización entre clases, cuidando que las cimas de las jerarquías de herencia estén formadas por clases abstractas, y desde un prisma conceptual se cumpla la relación tipo/subtipo entre los ancestros y sus descendientes [García and Marqués, 1998], o lo que es lo mismo el principio de sustitución de Liskov [Liskov, 1987]. Una buena aproximación es la combinación de esta relación con interfaces polimórficas vía delegación.

- **Composición.** Una clase puede utilizarse como parte de la representación de otra, siendo por tanto reutilizada. Esta es una aproximación muy flexible que denota una reutilización de *caja negra*, dado que los objetos no conocen los detalles internos de los otros.

- **Delegación.** La delegación es el mecanismo que permite que las relaciones de agregación se conviertan en una poderosa herramienta de reutilización, que puede llegar a ser incluso más potente y flexible que la herencia [Gamma et al., 1995].

  En la delegación dos objetos se encuentran involucrados en el manejo de una misma petición: *el objeto receptor* y *el objeto delegado*. El objeto receptor se pasa así mismo al delegado para permitir que la operación del delegado se refiera al receptor.

  La delegación tiene la ventaja de permitir el cambio de conducta del objeto receptor en tiempo de ejecución, si se ha establecido un correcto sistema de interfaces polimórficas. Esta propiedad se utiliza para el diseño y construcción de *frameworks*, de forma que una instancia de una clase del *framework* puede delegar en cualquier instancia de una clase cliente que implemente la interfaz establecida por el *framework*, adoptándose así el comportamiento del *framework* [D'Souza and Wills, 1999].

La principal diferencia entre la herencia y la delegación es que mientras que la primera clasifica objetos de acuerdo a su tipo o categoría, la segunda no define categorías de objetos, pero permite compartir estado y comportamiento entre objetos arbitrarios, admitiendo la definición incremental de los objetos [Stein, 1987].

- **Genericidad.** Las clases parametrizadas (*genéricas en Eiffel [Meyer, 1991] o plantillas en C++ [Stroustrup, 1997]*) permiten, mediante la instanciación de los parámetros de tipo apropiados, obtener nuevas clases adaptadas a diferentes contextos. Es por esto que la genericidad constituye un importante punto de apoyo en la reutilización de clases. En situaciones en las que se podría reutilizar una misma clase para tratar con elementos de diferentes dominios, sería deseable contar con la clase genérica adecuada.

Sin embargo, el diseño orientado a objetos toma su forma más avanzada en relación a la reutilización con la utilización de *patrones de diseño* y de *frameworks*.

Los patrones pueden considerarse como unidades de información destinadas a soportar, documentar y transmitir la experiencia en la resolución de problemas tipo que pueden aparecer en diferentes contextos. Cuando el contexto de los problemas a estudiar es el desarrollo de software se está dentro de los denominados patrones software.

El concepto de patrón software[7], más concretamente el concepto de patrón de diseño, se difunde gracias a los denominados Banda de los Cuatro (*The Gang of Four - o simplemente por GoF*) con su artículo [Gamma et al., 1993] y especialmente con la publicación del libro [Gamma et al., 1995].

Los patrones software, y en general la noción de patrón, constituyen una forma flexible y adecuada que favorece la reutilización de la experiencia embebida en los procesos que representan soluciones a problemas software, con independencia de su nivel de abstracción (*análisis, diseño o implementación*) [García, 1998].

Aunque los patrones se utilizan en diferentes contextos, aquí se presentan como mecanismos de reutilización informal de experiencia dentro de la tecnología de objetos, y desde este prisma se pueden definir como:

1. *Descripción en un formato fijo de cómo solucionar un cierto tipo de problemas [Reenskaug, 1996].*

[7] Realmente el uso del término patrón, con el significado que actualmente se le da en la Ingeniería del Software, y más concretamente en el área de la tecnología de objetos, se deriva de los trabajos del arquitecto Christopher Alexander, cuyos libros más afamados en este campo son [Alexander, 1964], [Alexander et al., 1975], [Alexander et al., 1977] y [Alexander, 1979].

2. *Unidad de información instructiva con nombre que captura la estructura esencial y la comprensión de una familia de soluciones exitosas probadas para un problema recurrente que ocurre dentro de un cierto contexto y de un sistema de fuerzas [Appleton, 1997].*

3. *Regla constituida por tres partes, la cual expresa una relación entre un cierto contexto, un cierto sistema de fuerzas que ocurren repetidamente en ese contexto, y una cierta configuración software que permite a estas fuerzas resolverse así mismas [Coplien, 1998].*

Con los *frameworks* se aplican los principios fundamentales de la orientación a objetos, pero en lugar de a la construcción de clases a la construcción de subsistemas o aplicaciones completas, de forma que el esfuerzo de desarrollar una nueva aplicación es proporcional a la diferencia de funcionalidad entre la aplicación particular y la proporcionada por el *framework*.

En contraste con las aproximaciones a la reutilización basadas en la composición de pequeños bloques almacenados en bibliotecas, los *frameworks* permiten un nivel de reutilización más alto al abstraer los elementos comunes a una familia de productos o aplicaciones, capturándolos en forma de estructuras y conceptos generales. Esto da lugar a un diseño genérico que será instanciado por cada aplicación particular que se cree empleando el *framework*.

En general un *framework orientado a objetos* se puede definir como "*un conjunto de clases diseñadas para trabajar juntas en la resolución de un problema o para ofrecer unas facilidades [Sparks et al., 1996]*".

Un *framework* puede verse como una biblioteca de clases que se construye sobre un uso sistemático y extensivo de la propiedad de ligadura tardía que se obtiene al utilizar interfaces polimórficas en los servicios de las clases.

En relación con los patrones, se puede decir que mientras que los patrones representan un conjunto de ideas que se pueden aplicar a diversas situaciones (*dicen cómo se puede resolver un problema*), un *framework* encierra una solución, incluyendo en su seno varios patrones diferentes.

Implementación

Las primeras experiencias con la reutilización del software se vieron limitadas a la reutilización de elementos de implementación, ya fuera código fuente o en formato binario.

La reutilización de elementos de implementación es un factor importante y, además de las tradicionales bibliotecas de funciones, los lenguajes de programación aportan diferentes métodos de modularización, empaquetamiento, genericidad, ocultación de la información, soporte de tipos abstractos de datos, mecanismos de extensión y compartición de código a través de la herencia...

Es de destacar el soporte que ofrecen varios sistemas operativos (*Windows 9x, Windows NT, OS/2, Unix...*) a la compartición de código gracias a utilidades de sistema que permiten la compilación separada y la creación de bibliotecas de enlace dinámico.

La aceptación obtenida primeramente por los controles **VBX** para Microsoft Visual BASIC[8] y posteriormente por los controles OLE2 (*OCX, actualmente ActiveX*) ha provocado la aparición en escena del denominado *Desarrollo Basado en Componentes* - **CBD** (*Component-Based Development*)[9] - donde las aplicaciones software son construidas *ensamblando* componentes provenientes de diferentes fuentes; de forma que los mismos componentes pueden ser escritos en diferentes lenguajes de programación y ejecutarse en distintas plataformas [Veryard, 1997]. La parte de la Ingeniería del Software encargada de velar por el desarrollo coherente de aplicaciones de calidad mediante el ensamblado de componentes es la **CBSE** (*Component-Based Software Engineering*) [Kozaczynski and Booch, 1998], [Brown and Wallnau, 1998].

Un apartado de especial importancia en el CBD es cuando se utilizan componentes realizados por terceras compañías, los denominados **COTS** (*Commercial Off-The-Shelf*) [Carney, 1997], [Oberndorf, 1998], [Voas, 1998], constituyendo lo que se viene a denominar *software comercialmente disponible* - **CAS** (*Commercially Available Software*). La utilización de los COTS conlleva unos factores de seguridad y previsión de riesgos más acentuados que en el desarrollo de las aplicaciones convencionales [Lindqvist and Jonsson, 1998], así como unos problemas de integración nada triviales [Boehm and Abts, 1999].

1.1.8 Reutilización y Orientación a Objetos

Nadie es ajeno a que la orientación a objetos, junto con el web, son las dos tecnologías que mayor índice de impacto han tenido en los últimos años de la década de los noventa, y que más prometedor futuro se les augura para el siglo que se avecina; opinión que suscriben importantes expertos de relevancia internacional, entre otros Grady Booch [Booch, 1994]

[8] Autores como Edward Yourdon creen que el beneficio real de entornos como VB o Powerbuilder radican más en la facilidad que ofrecen para la reutilización que e el propio paradigma visual [Yourdon, 1996].

[9] También conocido como Programación Orientada a Componentes - **COP** (*Component-Oriented Programming*) [Jazayeri, 1995], [Szyperski, 1995] - o como Componentware.

o Steve Jobs [Wolf, 1997].

Por otra parte, la reutilización se cita de forma sistemática como uno de los principales objetivos de la tecnología de objetos[10]. Esto se constata en diversos estudios realizados en Estados Unidos, Japón y Europa de los cuales se pueden obtener interesantes indicadores.

Sin embargo, en contra de lo que pueda parecer, la reutilización es una disciplina por sí misma, ortogonal a cualquier paradigma de desarrollo. Aunque no deja de ser cierto que el binomio reutilización - orientación a objetos da lugar a una relación simbiótica entre estas tecnologías. Por un lado la orientación a objetos aporta un modelo de referencia con una gran cantidad de elementos que favorecen la reutilización (*abstracción, encapsulamiento, jerarquías de herencia, jerarquías de composición, delegación...*). Por otro lado, la reutilización se convierte en un punto esencial para que, con el uso de la tecnología de objetos, se puedan conseguir el resto de los beneficios potenciales.

No obstante, esta relación simbiótica podría expresarse como una relación de dependencia de la orientación a objetos con respecto a la reutilización, afirmación que se justifica con las siguientes razones:

- Por el hecho de utilizar técnicas orientadas a objetos no se está reutilizando automáticamente [Griss, 1995a]. Se necesita un esfuerzo extra para obtener la potencia que el modelo objeto puede ofrecer en pro de la reutilización del software [Johnson and Foote, 1988], [Johnson and Russo, 1991], [Meyer, 1994], [Meyer, 1997], [García et al., 1997a].

- Los proyectos de gran tamaño realizados con técnicas de objetos pueden volverse incontrolables si no se realizan sobre la base de la reutilización [McClure, 1996].

- La simple adopción de la tecnología de objetos, cuidando el desarrollo para reutilización, no garantiza una reutilización sistemática efectiva del software. Para lograr una reutilización a gran escala se deben unir factores tecnológicos (*métodos orientados a objetos con soporte para la reutilización [Griss, 1996b]*) con factores organizativos (*soporte de la dirección, cambios en la organización y cambios en el personal [Griss and Wosser, 1995]*).

[10] Junto con la reducción del tiempo de desarrollo, el aumento de calidad, la mejora de la productividad y la facilidad de mantenimiento de las aplicaciones [McClure, 1997].

1.2 Problemas en la Reutilización del Software

El trabajo realizado y los avances logrados dentro de la investigación en el campo de la reutilización del software han sido considerables en los diferentes aspectos de esta disciplina [Poulin, 1999]; de hecho, a estas alturas no se discute la necesidad de contar con la reutilización dentro de un proceso de desarrollo de software maduro[11].

Pero la realidad es que, pese a los beneficios potenciales que ofrece la reutilización del software, no hay una correspondencia directa entre los esfuerzos de investigación y su empleo real en los procesos de desarrollo del software [Glass, 1998], [Griss, 1999b]. Este es sin duda alguna el mayor de los problemas que sufre la reutilización; problema que viene motivado por un cúmulo de circunstancias que se pueden resumir en los siguientes puntos:

- Existe un desconocimiento general en el tema que ha dado lugar a diversos mitos e inhibidores de la reutilización (*falta de inceptivos para reutilizar, desconfianza a lo desarrollado por otros, mentalidad artesana en el desarrollo del software, falta de formación...*), los cuales han sido objeto de diversas ponencias y discusiones por parte de los mayores expertos en reutilización [Tracz, 1988b], [Tracz, 1988c], [Frakes et al., 1991], [Wentzel, 1994], [Basili, 1994], [Tracz, 1994], [Wasmund, 1994].

- El concepto de reutilización más difundido está centrada en el código fuente [Johnson, 1995], donde el "*cortar y pegar*" es la técnica más difundida. Esto lleva a que si se toma un modelo de madurez de la reutilización que la clasifique según su uso en **oportunista, integrada, comprometida, anticipadora** [Favaro, 1999], se tendría que la mayoría de las organizaciones practicarían una forma oportunista de reutilización.

- La incorporación de la reutilización del software cambia de forma radical el proceso de desarrollo del software, tanto en los roles de las personas involucradas, como en los modelos de desarrollo, métodos y tecnología empleados [Griss and Wentzel, 1994].

- La reutilización no es un fin en sí misma, debe justificarse en los objetivos de la organización, donde los factores económicos a medio/largo plazo son decisivos para seguir adelante con un plan de reutilización [Frakes and Fox, 1995].

[11] Como dato anecdótico, corroborando ese camino de éxito y de madurez que se le vaticinaba a la reutilización, Rubén Prieto-Díaz predecía en la Conferencia Internacional sobre Reutilización **ICSR-3**, celebrada en Río de Janeiro en 1994, que hacia el año 2000 dejaría de existir (*en el sentido de que se integraría dentro del proceso general de la Ingeniería del Software*), siendo precisamente su desaparición el último éxito de la reutilización del software [Prieto-Díaz, 1994].

- La creación de un programa de reutilización sistemática dentro de una organización supone una inversión a medio/largo plazo que no siempre se está dispuesto a afrontar.
- Existencia de un importante salto entre **el estado del arte** y **el estado de la práctica** en el campo de la reutilización del software [Guerrieri, 1999], [Basili et al., 1999], responsable de que la mayoría de los avances realizados en tecnología, métodos, procesos y factores de organizaciones queden como trabajos teóricos o en *prototipos de laboratorio*.

Así, pues se debe seguir avanzando en el estado del arte de la reutilización, tanto en el ámbito técnico como en el organizativo, siendo fundamental la realización de un esfuerzo para que estos avances se traduzcan en beneficios prácticos y no se tomen sólo como meras satisfacciones intelectuales.

1.3 Objetivos

En un intento de dar respuesta al problema de reutilización sistemática del software se propone la realización de un trabajo de tesis doctoral que establezca un **modelo de reutilización sistemática** soportado por elementos reutilizables de grano grueso, los MECANOS.

Para ello se plantea, en primer término, la elaboración de un modelo de elemento reutilizable de grano grueso que proporcione la estructura o composición de estos elementos reutilizables que reciben el nombre de mecanos. Este modelo hace un especial hincapié en una serie de principios básicos, destacando:

- La potenciación del aumento del nivel de abstracción del proceso reutilización.
- El soporte simultáneo de diferentes niveles de abstracción.
- El soporte para la trazabilidad.

En segundo lugar se define el proceso de construcción de los mecanos, distinguiéndose entre *construcción manual de mecanos*, realizada por el desarrollador para reutilización y una *composición automática de mecanos*, guiada por el desarrollador con reutilización.

En tercer lugar, teniendo en cuenta que la disponibilidad de un repositorio repleto de elementos reutilizables, mecanos en este caso, no es suficiente por sí mismo para que la reutilización sistemática sea un hecho, se propone un proceso de desarrollo que dé cabida

a la reutilización del software, otorgando un sentido a los mecanos para la organización, el cual, atendiendo a las tendencias actuales, está basado en líneas de producto, por ser éste el enfoque que mayores beneficios aporta.

En cuarto y último lugar se busca asegurar la calidad del producto final que se desarrolle con reutilización. El camino para que esto se logre comienza por establecer un proceso de cualificación de los elementos que se reutilizan, guiado por un plan de calidad adecuado.

En resumen, el modelo de reutilización sistemática del software propuesto consta de tres apartados diferenciados. El primero centrado en los aspectos tecnológicos y de soporte de la reutilización, el **modelo técnico de reutilización**. El segundo establece el proceso de reutilización, con las implicaciones oportunas en la organización de los recursos involucrados, el **modelo de proceso de reutilización**. El tercero, vela por la certificación de la calidad de los elementos reutilizables que pueblan el repositorio, el **modelo de cualificación de reutilización**.

1.4 Contexto del Trabajo

Este trabajo de tesis doctoral continua con la línea de investigación sobre reutilización del software iniciada en el Departamento de Informática de la Universidad de Valladolid por el Dr. José Manuel Marqués Corral [Marqués et al., 1994], [Marqués, 1995], [Marqués, 1996].

Es de destacar que el presente trabajo ha sido realizado al amparo del proyecto coordinado **MENHIR**[12] (*Modelos, Entornos y Nuevas Herramientas para la Ingeniería de Requisitos*), que tiene como objetivo el desarrollo de un entorno de producción de software de calidad (*correcto, reutilizable y mantenible*) que abarque todo el ciclo de vida haciendo uso de los formalismos necesarios para soportar los sucesivos modelos que constituyen el producto software en sus tres aspectos: estado, comportamiento y comunicación [Grupo MENHIR, 1998].

En concreto esta tesis constituye el núcleo teórico del subproyecto **MECANO** dentro del proyecto **MENHIR**, y del cual se encarga el grupo **GIRO** (*Grupo de Investigación en Reutilización y Orientación a Objetos*) de la Universidad de Valladolid, liderado por el Dr. José Manuel Marqués Corral, y cuyo objetivo se resume en: "*la definición y diseño de 'mecanos' reutilizables como soporte a la construcción rápida de aplicaciones*".

[12] Proyecto subvencionado por CICYT-TIC97-0593-C05-01, y en el que confluyen grupos de investigación de seis universidades españolas (*Castilla-La Mancha, Granada, Murcia, Sevilla, Politècnica de Valencia y Valladolid*) y que tiene como investigador principal del mismo al Dr. Isidro Ramos Salavert de la Universidad Politècnica de Valencia.

1.5 Presentación General del Resto de los Capítulos

Como se indica con anterioridad el centro de esta propuesta de tesis doctoral lo constituye un modelo de elemento software complejo de reutilización, que define la construcción de unos elementos reutilizables de grano grueso, compuestos de una serie de elementos reutilizables de grano fino que reflejan los diferentes estados de abstracción por los que evoluciona un desarrollo software. El capítulo dos aborda el estudio de los elementos complejos (*de grano grueso*) de reutilización que han surgido en otros trabajos relacionados. Además, partiendo del hecho de que se optado por definir una estructura de reutilización propia, se analizan los elementos necesarios para su definición, haciendo un estudio crítico de lo existente en estos campos. Así, este capítulo se convierte en el estado del arte de las estructuras o elementos reutilizables.

En el tercer capítulo se define la estructura del elemento reutilizable sobre el que gira toda la problemática tratada en esta propuesta de tesis doctoral, el Modelo de Mecano. Este modelo engloba dos partes relacionadas pero que es importante distinguir. Por un lado el modelo del componente reutilizable propiamente dicho, el *mecano*, que es un elemento software de grano grueso compuesto por un conjunto de *assets*, o elementos reutilizables de grano fino, relacionados entre sí. Por otra parte el modelo de repositorio, que además de acoger el modelo de componente, establece todos los elementos de información que se necesitan almacenar para ofrecer un servicio de reutilización basado en mecanos, con todas las garantías de compleción y calidad. Tanto el modelo de componente como el modelo de repositorio se definen mediante técnicas semi-formales, utilizando un enfoque y un lenguaje de modelado (*UML 1.x*) orientado a objetos, y razonando cada una de las decisiones que se han tomado para llevar a cabo su definición (*partiendo del estado del arte realizado en el Capítulo 2*). Por último, se proporciona una segunda definición del modelo de componente reutilizable, ésta formal soportada por la teoría de grafos.

El cuarto capítulo aborda el proceso de construcción de mecanos, distinguiéndose dos situaciones. La primera, encuadrada en el desarrollo para reutilización, explica cómo construir mecanos mediante un enfoque compositivo puro, guiado de forma "*manual*" por un desarrollador para reutilización. La segunda aborda el caso en el que, ante la ausencia de mecanos que satisfagan las necesidades del reutilizador, se construyen mecanos mediante un proceso de composición o extracción automática. En ambos casos ha de garantizarse que el proceso de construcción da como resultado un *mecano bien formado*, lo cual se garantiza a través de la definición de una gramática soportada por el modelo formal establecido en el *Capítulo 3*. La presencia de estas dos variantes supone un enfoque mixto generativo/compositivo.

En el capítulo cinco se presenta el marco general de referencia para la reutilización del software mediante un modelo de reutilización basado en mecanos. En este modelo se distinguen tres áreas de acción principales (*técnica, proceso y calidad*) que agrupan todo el enfoque de reutilización que se deriva de la construcción y gestión de los mecanos.

El sexto capítulo recoge las conclusiones y las líneas de trabajo que se abren a partir de los resultados obtenidos.

Como complemento a los capítulos anteriores, se incluye un apéndice con las aplicaciones prácticas que se han derivado de los resultados teóricos de esta propuesta de tesis doctoral. Dichas experiencias prácticas reflejan de una forma fiel la evolución del trabajo, desde la adaptación de herramientas existentes para crear un primer prototipo, hasta la construcción del prototipo inicial de un repositorio en el que se introduce de forma nativa el concepto de mecano.

Capítulo 2

Elementos Software Reutilizables de Grano Grueso

El desarrollo de nuevas aplicaciones mediante la composición de elementos reutilizables de grano fino (*assets*) es una de las formas de reutilización de mayor difusión. Cuando se hace uso de este enfoque de reutilización se debe procurar aumentar el nivel de abstracción de los assets a reutilizar porque, aún pagando el precio de su mayor dificultad de reutilización, los beneficios que reporta son mucho mayores [Cybulski, 1997].

Sin embargo, esta forma de reutilizar no está exenta de problemas. Así, la construcción de aplicaciones mediante la reutilización de assets de origen heterogéneo es sumamente complicada y con pocas garantías de éxito; esto es lo que se conoce como la *la metáfora de los bloques Lego* [Ran, 1999], y es la aproximación que ofrecen la mayoría de las bibliotecas de reutilización o repositorios actualmente existentes.

Ante esta perspectiva parece necesario buscar unos elementos reutilizables de mayor granularidad que permitan tanto el aumento del nivel de abstracción involucrado en el proceso de reutilización del software, como el soporte simultáneo de diversos niveles de abstracción mediante la inclusión de un sistema de assets de grano fino estrechamente relacionados por su origen en un esfuerzo de desarrollo común.

Además, no debe perderse de vista otras interesantes ventajas de estos elementos reutilizables de grano grueso. Así, desde un punto de vista técnico se tiene un soporte intrínseco de la trazabilidad entre los diferentes assets de grano fino. Por otra parte, desde una perspectiva de organización, se establece un elemento sólido sobre el que construir un modelo de reutilización desde el que dar soporte a la reutilización sistemática del software o reu-

tilización a gran escala [Henry and Faller, 1995], [Griss, 1999a], con una transición suave a la aproximación de **líneas de producto**, actualmente defendidas por muchos autores como la forma más adecuada de conseguir altas cuotas de reutilización [Clements, 1997b], [Maymir-Ducharme, 1997], [Griss et al., 1998].

Este capítulo se organiza en dos secciones. En la primera de ellas se realiza un análisis crítico de aquellas estructuras complejas de reutilización que definen los modelos de elementos software reutilizables de grano grueso que, habiendo sido desarrollados en otros importantes trabajos de investigación en el campo de la reutilización sistemática del software, sirven como base a la propuesta de tesis doctoral que aquí se desarrolla. Posteriormente, en la sección dos, se hace un estudio del estado del arte de aquellas áreas que intervienen en la definición de la estructura compleja de reutilización que sirve como modelo para la construcción de los elementos reutilizables de grano grueso propuestos, **los mecanos**.

2.1 Estado del Arte de las Estructuras Complejas de Reutilización

Obtener una representación de un elemento reutilizable, esto es su estructura, que de soporte a una reutilización sistemática a lo largo del ciclo de vida del software (*soporte simultáneo de diferentes niveles de abstracción*) es un problema de la reutilización [DoD, 1995] que repercute tanto en los aspectos técnicos como organizativos de ésta.

En la bibliografía se encuentran diversas aproximaciones a elementos reutilizables de grano grueso, aunque la mayoría de ellas se presentan como meras representaciones arquitectónicas para las líneas de producto, y más que preocuparse por la definición o modelo de la estructura compleja de reutilización, se centran en aspectos metodológicos u organizativos, quedando la definición de la estructura en un segundo plano, normalmente encuadrada dentro del contexto general expuesto en el trabajo.

Es interesante destacar que las líneas de producto son una forma de organizar un enfoque de reutilización sistemática dentro de una institución. Esto conlleva múltiples decisiones en torno al establecimiento de un plan de reutilización dentro del proceso de desarrollo de la organización. Entre estas decisiones se debe establecer la estructura compleja de reutilización que de soporte a los diferentes assets que definen tanto a la línea de producto (*requisitos, variantes, arquitectura, plan de implementación... [Cohen et al., 1996]*) como a los productos en sí.

Entre estas estructuras cabe destacar algunos ejemplos especialmente representativos.

Así, **ODM 2.0** habla de una colección de assets, dentro de un dominio, que se presentan como un todo, determinando una arquitectura única para cada tipo de problema tratado. Esta estructura recibe el nombre de **asset base** [Simos et al., 1996]. La noción de asset base, como el núcleo central compuesto por assets desde el que se crean los productos de la línea de producto o de la familia de productos, se repite constantemente en la bibliografía relacionada con las líneas de producto, referenciándose también como **plataforma** [Bass et al., 1998].

En **RSEB** [Jacobson et al., 1997] se presenta un elemento de grano grueso, denominado **sistema de componentes** que representa una agregación de assets de grano fino relacionados entre sí para ofrecer una funcionalidad mayor de la que pueda dar un asset individual, buscando una facilidad de manejo por parte del desarrollador con reutilización. Un sistema de componentes presenta sólo un subconjunto de la información que posee a través de **fachadas**; es decir, cada fachada presenta aquellos aspectos del sistema de componentes que necesita el desarrollador con reutilización.

Sin embargo, las dos estructuras de reutilización complejas más relevantes son los **Application Frames** del proyecto ESPRIT **ITHACA** y los **kits específicos de un dominio**.

2.1.1 Application Frames

El proyecto **EEC-SPRIT II ITHACA** (*Integrated Toolkit for Highly Advanced Computers Applications*) [Ader et al., 1990] se llevó a cabo entre los años 1989 y 1992 con el objetivo de establecer un entorno de desarrollo software soportado en dos bases fundamentales: la orientación a objetos y la reutilización.

El centro de este entorno es el **SIB** (*Software Information Base*) [Constantopoulos et al., 1992], [Constantopoulos and Dörr, 1995], lugar (*repositorio*) donde se almacenan y publican para su reutilización objetos semánticos, objetos de diseño y objetos de implementación; esto es, descripciones del software en diferentes niveles de abstracción (*requisitos, diseño y código*). En torno al repositorio hay una serie de herramientas para soportar tanto la Ingeniería de Dominio como la Ingeniería de Aplicación [Fugini et al., 1992], [de Mey and Nierstrasz, 1993].

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 la documentación [Bellinzona et al., 1993].

Por tanto, en este repositorio se almacenan assets de diferentes niveles de abstracción, desde descripciones de requisitos a descripciones de implementación. Esto significa que especificaciones software de más alto nivel pueden ser reutilizadas directamente, pero también pueden servir como índices a assets definidos en niveles menor abstracción [Bellinzona et al., 1995].

Si sólo existiesen descripciones de elementos reutilizables clasificados en diversos niveles de abstracción, aun estando enlazados, su reutilización práctica sería caótica. Por este motivo se utiliza un elemento de mayor granularidad para organizar el SIB, el *application frame* o simplemente *frame*.

Un *application frame* representa un sistema completo o una familia de sistemas, que tiene al menos una implementación y opcionalmente descripciones de diseño y de requisitos. Los *application frames* pueden catalogarse en genéricos y específicos [Constantopoulos et al., 1992].

El objetivo del desarrollador para reutilización es producir *application frames* genéricos[1] - **GAFs** (*Generic Application Frames*) - que sirvan como plantillas para que los desarrolladores con reutilización puedan trabajar.

Cuando un desarrollador con reutilización selecciona un GAF y lo completa para producir una aplicación específica, está produciendo un *application frame* específico - **SAF** (*Specific Application Frame*) - que describe un sistema completo e incluye exactamente una implementación.

En resumen, ITHACA establece una estructura compleja de reutilización, el *application frame*, que origina elementos reutilizables de grano grueso y soporte simultáneo de diferentes niveles de abstracción, con un proceso de reutilización que engloba tanto la Ingeniería de Dominio como la Ingeniería de Aplicación; pudiendo ver en la conjunción de los GAFs y los SAFs una forma de primitiva de soporte para el concepto de línea de producto.

Sin embargo, los *application frames* no son válidos para el objetivo perseguido por las siguientes razones:

- *El único asset de grano fino considerado (a excepción de la documentación) es la descripción de clases en diferentes niveles de abstracción. Esto es debido a su estrecha relación con la tecnología de objetos, pero que lo hace inviable bajo una perspectiva de desarrollo que sea independiente del paradigma.*

[1] Comenzando con *frames* vacíos o con uno o varios *frames* existentes, se desarrolla un **GAF** que encapsule toda la información sobre una familia de aplicaciones, desde los requisitos a assets de implementación genéricos.

- *Las descripciones de las clases son dependientes del entorno de desarrollo definido en el proyecto (esto es, del conjunto de herramientas que envuelven al SIB). Lo cual es contrario al establecimiento de un en enfoque de reutilización en el que el elemento reutilizable sea cualquier producto construido bajo un enfoque metodológico, sea cual sea éste, no estando ligada su creación a ninguna herramienta específica.*

2.1.2 Kits Específicos de un Dominio

Como parte de la iniciativa de Hewlett-Packard para la inclusión de la reutilización sistemática dentro su proceso de construcción de software, se constituye el **Programa de Reutilización Coorporativo** liderado por Martin L. Griss en el inicio de la década de los 90's.

En los momentos iniciales se buscaba que los desarrolladores de aplicaciones contaran con unas colecciones de elementos software extensibles, que se pudieran combinar con generadores de aplicaciones específicas de un dominio. Había nacido el concepto de **kit específico de un dominio** [Griss, 1991], que se puede definir como *el 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 [Griss, 1993a].*

De la definición se deriva que un *kit* es un elemento reutilizable complejo que involucra diferentes niveles de abstracción dentro sus componentes.

La evolución y madurez del concepto de *kit* conduce hacia los *kits híbridos específicos de un dominio* [Griss, 1993b], [Griss and Wentzel, 1994], [Griss and Wentzel, 1995]. Esta estructura es especialmente interesante porque representa una aproximación mixta de reutilización por composición y por generación.

Un *kit* híbrido típico (*genérico*) estará compuesto por los siguientes elementos, como se mostraba en la figura 1.1:

- *Un conjunto de assets*, que están documentados, probados y empaquetados de forma adecuada: funciones, clases, plantillas...
- *Un framework*, que permita combinar los assets compatibles con código o mediante la utilización de lenguajes "*pegamento*", permitiendo añadir productos externos al *kit*.
- *Un lenguaje "pegamento"*, que permita unir componentes y añadir funcionalidad extra. Debe ser de propósito general, flexible, basado en *scripts* o específico del dominio.

- *Aplicaciones genéricas de ejemplo*, que sirvan como plantillas en las que sólo se tengan que fijar una serie de parámetros y métodos, para que enlazando con los otros elementos del *kit* se obtenga rápidamente un prototipo, siguiendo un ciclo de vida basado en prototipos [Fischer, 1987], [Johnson et al., 1993].

- *Un entorno de desarrollo*, que ofrezca las utilidades de desarrollo necesarias: *navegador de bibliotecas, editor, compilador, generador, depurador...*

- *Un entorno de ejecución*, donde configurar y ejecutar las aplicaciones.

Los *kits* híbridos se encuentran entre las bibliotecas de componentes y otros elementos más altamente configurables, como son los generadores de aplicaciones.

El concepto de *kit* híbrido es perfectamente válido tanto para la construcción de software bajo el paradigma objetual [Griss and Kessler, 1996], como para la construcción de aplicaciones tradicionales. Sin embargo, las aplicaciones prácticas de este concepto han derivado hacia entornos visuales y **RAD** (*Rapid Application Development*), siendo la prueba más evidente de este hecho la concepción de Visual Basic como el *kit* ideal [Griss, 1995b], [Griss and Kessler, 1996].

Como conclusión, se puede decir que los *kits híbridos específicos de un dominio* ofrecen una aproximación a la reutilización con un enfoque multinivel de abstracción y, lo que es más importante, una combinación de técnicas de composición con técnicas de generación.

Sin embargo, esta aproximación se aleja del enfoque de trabajo propuesto, principalmente por las siguientes razones:

- No define una estructura concreta, sino que identifica un conjunto de elementos que deben estar presentes en la formación del *kit*.

- Establece un enfoque de reutilización sistemática que debe ser implementado en cada entorno de desarrollo concreto, como por ejemplo Visual Basic. Esto hace que se pierda la idea de un modelo de reutilización general independiente de las herramientas utilizadas por los desarrolladores para y con reutilización.

- Aunque en la definición de *kit* se menciona la inclusión de elementos de todo el ciclo de vida, en la práctica está orientado a la reutilización de assets de implementación, especialmente código y componentes binarios. En consecuencia se pasa a un segundo plano la reutilización de assets de análisis y diseño, porque la información del dominio queda embebida en los generadores y el diseño en los *frameworks* y bibliotecas predefinidas utilizadas.

2.1.3 Influencias de las Estructuras Complejas Estudiadas

Como se ha puesto de manifiesto, ninguna de las estructuras estudiadas se ajusta por completo a los objetivos marcados en el presente trabajo; no pudiéndose negar, sin embargo, su inevitable influencia en la definición de los mecanos como estructuras complejas de reutilización.

Así, en cuanto a la estructura de reutilización que defina los elementos software reutilizables, la aproximación más cercana son los sistemas de componentes de RSEB, porque aparecen assets de todo el ciclo de vida empaquetados en un verdadero elemento reutilizable de grano grueso que puede ser recuperado gracias al mecanismo de las fachadas que publica la información relevante para el desarrollador con reutilización. Sin embargo, hay que buscar un enfoque más abierto que permita trabajar con un amplio espectro de elementos de grano fino para su constitución, haciendo especial hincapié en la relación existente entre estos elementos.

En cuanto al entorno operativo de la reutilización, en el que los elementos son manejados, debe huirse de los sistemas cerrados, al estilo de ITHACA, porque tienen el grave inconveniente de tener un ciclo de vida ligado al laboratorio donde se realiza el proyecto de turno, siendo difícil su transferencia a la realidad cotidiana de las empresas.

Tampoco, es adecuado un enfoque tan ligado a productos comerciales como en el que han derivado los *kits* híbridos, porque se está cerrando la puerta a muchos potenciales beneficiarios de la reutilización que no compartan el entorno escogido.

En este sentido se debe buscar un entorno operativo de la reutilización que sea ortogonal a los métodos y herramientas de desarrollo de la empresa que lo vaya a utilizar, siendo actualmente una realidad que esté sea accesible a través de servicios web, ya sea de una forma más o menos altruista en Internet o de forma privada y controlada en una intranet [Browne and Moore, 1997], [Trump, 1997].

En lo que respecta a la forma de construir los elementos reutilizables, se sigue principalmente un camino compositivo, como marca ITHACA, pero buscando una mayor flexibilidad es interesante contemplar un enfoque híbrido, como se propone en los *kits* híbridos, y la forma elegida se deriva de una composición automática de los elementos reutilizables dentro de las actividades propias del desarrollo con reutilización, teniendo como trasfondo la filosofía de *componentes entran, sistemas salen* propugnada por **CARDS** (*Comprehensive Approach to Reusable Defense Software*) [Wallnau, 1992], [Petracca and Bock, 1994].

Por último, y en relación a los aspectos organizativos, se debe buscar un acercamiento

a las líneas de producto, de forma que los elementos reutilizables de grano grueso sirvan tanto para establecer un asset base, como para representar los productos derivados de dicho asset base.

Todas estas influencias se resumen en la figura 2.1.

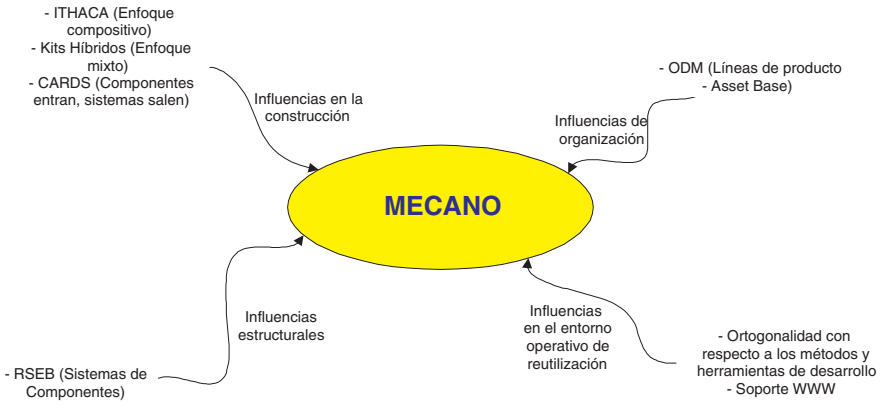

Figura 2.1: Influencias en los mecanos.

2.2 Elementos de una Estructura Compleja de Reutilización

A la hora de definir la estructura o el modelo del elemento software reutilizable de grano grueso, con soporte simultáneo de diferentes niveles de abstracción, que sirve como núcleo al modelo de reutilización que se presenta en esta tesis, se deben tomar una serie de decisiones capitales en cuanto a los elementos estructurales que definen dicho modelo; estos son: *los niveles de abstracción considerados, el modelo de componente de grano fino (asset)* y *las relaciones existentes entre dichos assets.*

En esta sección se van a presentar las contribuciones más relevantes en cada área, buscando que este estudio aporte una realimentación a la definición de la estructura propuesta.

2.2.1 Niveles 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 bibliografía se tienen numerosos ejemplos de criterios de clasificación de assets en

relación al atributo de la abstracción de los mismos.

Así, Charles W. Krueger [Krueger, 1992] define una taxonomía de la información reutilizable basándose en niveles de abstracción, donde 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, elementos 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: elementos 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 está 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., 1997b] 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 bibliografía es atendiendo a una clasificación muy simple según la fase del desarrollo y/o el nivel de abstracción en el que se produce o se reutiliza el conocimiento. Este doble criterio establece la reutilización según tres niveles: reutilización de código, reutilización de diseños y reutilización de especificaciones. Este enfoque, sumamente difundido, ya fue introducido al hablar de ITHACA [Constantopoulos et al., 1992], [Bellinzona et al., 1993], pero está presente en muchos otros trabajos, por ejemplo en [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 elemento reutilizable de grano fino como *la etapa*

del ciclo vital de desarrollo software, caracterizada por la cercanía al nivel del dominio del problema o al nivel del dominio de la solución de los elementos software generados en dicha etapa.

2.2.2 Modelos de Asset

Un aspecto de especial relevancia en la definición de la estructura compleja de reutilización es la representación de los elementos de grano fino que la componen. Esta representación, o modelo de elemento reutilizable de grano fino, tiene como objetivo describir la información que se necesita almacenar junto al asset con la finalidad de facilitar la utilización del asset dentro de un proceso de Ingeniería de Aplicación.

Modelo de Elemento Reutilizable según REBOOT

Dentro del proyecto **REBOOT** [Karlsson, 1995], [SER Consortium, 1996], se aporta un modelo de elemento software reutilizable que sirve como marco de referencia para la implementación práctica de algunos repositorios[2]. Este modelo se presenta en la figura 2.2[3].

El modelo de REBOOT no presenta una definición concreta de los campos de información que deben acompañar al asset en el repositorio, de hecho se limita a indicar que tipo de información es la que debe aparecer:

- ***La clasificación*** de un asset es la información que se guarda junto con el asset y que sirve de ayuda para su identificación y recuperación. 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 un conjunto de métricas. También pueden tener cabida comentarios, problemas surgidos al reutilizar el asset, soluciones a dichos problemas... De

[2]Según el enfoque metodológico establecido en REBOOT, para hacer factible la reutilización, los assets deben estar almacenados en un repositorio junto con la información necesaria que permita su recuperación y la evaluación de su ajuste a los requisitos buscados, además de facilitar su adaptación e integración en el sistema global si fuera necesario.

[3]Este modelo se ha adaptado utilizando la notación UML 1.x [OMG, 1998], [OMG, 1999], [Booch et al., 1999], en lugar de la representación mediante un diagrama entidad/relación con el que originalmente se presenta en [Karlsson, 1995].

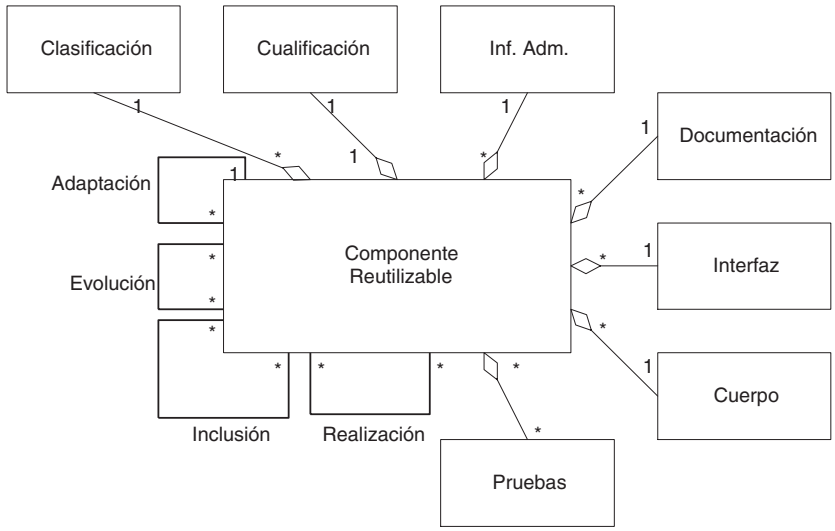

Figura 2.2: Modelo del elemento reutilizable definido en REBOOT [Karlsson, 1995].

esta forma 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 (*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*), 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.

- ***La documentación*** es esencial en la reutilización de un asset. 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.

- ***La interfaz del asset*** describe la forma de comunicarse con el asset, es decir, qué operaciones ofrece, qué parámetros toma y qué requiere de su entorno. Por su parte, el cuerpo del asset es la descripción de como trabaja el asset internamente. Se tiene 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, ya que los assets 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 de varios a varios, ya que un asset puede tener varios juegos de prueba, y un juego 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 subsiguientes. La relación de realización es una relación varios a varios entre assets, debido a que una especificación puede estar asociada a muchas soluciones de bajo nivel, y viceversa.

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

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

- ***La relación de adaptación*** muestra las diferentes adaptaciones sufridas por 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.

Después de la breve presentación realizada del modelo de asset propuesto por REBOOT, se resumen los aspectos más relevantes de este apartado con relación a los intereses del presente trabajo.

- 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. Esto se fijará a posteriori en cada una de las implementaciones de repositorio que tomen a REBOOT como referencia.

- Aparece la abstracción como un atributo para la clasificación de los assets. Así, se presenta una relación entre assets clasificados en niveles de abstracción diferentes

(*relación de realización*). No obstante, la definición que se hace de esta relación es demasiado general, carente de cualquier connotación semántica, presentándola como un simple enlace entre assets.

- Es especialmente pobre en cuanto al modelado de las relaciones semánticas entre los assets clasificados en un mismo nivel de abstracción, contemplado exclusivamente la agregación y el versionado de assets.
- El modelo de asset propuesto por REBOOT abusa de las cardinalidades múltiples entre el asset y los componentes de información. Por motivos de claridad, se debería buscar relaciones uno a uno entre el asset y los elementos de información existentes.
- Algunos de los componentes de información propuestos por REBOOT pueden ser considerados como elementos reutilizables por sí mismos, asociándolos a otros assets, por ejemplo las pruebas y las documentaciones.
- Basado en REBOOT existe una implementación de un motor de repositorios comercial, EUROWARE [SER Consortium, 1996].

Modelo de Elemento Reutilizable Propuesto en EUROWARE

El proyecto ESPRIT #8947, **EUROWARE** [SER Consortium, 1996], [Sema Group, 1996], [Villa, 1997], 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 modelo de datos de EUROWARE presenta una serie de entidades de las cuales, para el fin que se persigue en el presente apartado, sólo se presentará la entidad *Asset*.

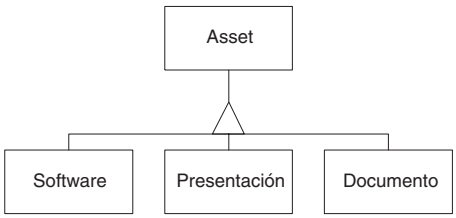

Figura 2.3: Jerarquía de assets en EUROWARE.

Asset es una clase abstracta que se especializa en tres tipos de assets (software, presentación y documento), y que recoge los elementos comunes a los subtipos de asset. (*Ver figura 2.3*).

Un asset genérico tiene un conjunto de atributos de carácter general que serán heredados por los tipos específicos de assets. Dichos atributos son:

- **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 el repositorio. Este atributo es asignado por el servidor.
- **Fecha de última modificación:** La fecha de última modificación del asset.

Un asset software amplia los atributos generales con atributos de información relativa al entorno de creación y explotación de los assets (*entorno, sistema operativo, plataforma, compilador...*).

Un asset de presentación añade a los atributos heredados atributos de información referente a la presentación en sí (*tipo de presentación, número de diapositivas...*).

Por último un asset documento hereda todos los atributos de asset y, además, añade los siguientes: *formato, nivel técnico y audiencia.*

Una descripción más detallada de los atributos de los tipos de assets soportados por EUROWARE, así como de su significado, se encuentra en [García et al., 1998c].

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** con instancias 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** con instancias de **T1**.

Existen varias relaciones en EUROWARE, siendo las más importantes:

- *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 se tiene información relativa a la persona que lo insertó, y la cual se entiende como su responsable (*nombre, número de teléfono, número de fax, correo electrónico...*).

A modo de resumen, sobre EUROWARE se puede concluir que:

- EUROWARE implementa un subconjunto de las propuestas expresadas en el modelo de asset de REBOOT.
- 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.
- La jerarquía establecida por EUROWARE es demasiado simplista, pudiéndose llegar a catalogar incluso de artificiosa. Por una parte, el tipo *software* es algo demasiado genérico que requiere 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 está 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.

El Modelo de Elemento Reutilizable Propuesto en el RIB

El repositorio **RIB** (*Repository In a Box*) de **NHSE** (*National HPCC Software Exchange*) [Browne et al., 1995], es un conjunto de herramientas para la creación de repositorios software que puedan compartir información a través de Internet. RIB presenta una extensión del modelo **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], [Browne and Moore, 1997].

El modelo BIDM es un modelo objeto compuesto básicamente por cuatro clases y sus relaciones: **Asset, Elemento, Biblioteca** y **Organización**. Un asset es una entidad reutilizable generada en el ciclo de vida de desarrollo. Un elemento es un fichero. Una biblioteca está compuesta de assets. Y por último, una organización puede ser una persona,

una compañía, un grupo de investigación... que crea y gestiona un asset o una biblioteca [NHSE, 1997b].

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

En la figura 2.4 se presenta el modelo BIDM, pero usando UML en lugar de la notación de OMT [Rumbaugh et al., 1991] originalmente utilizada en [NHSE, 1997b].

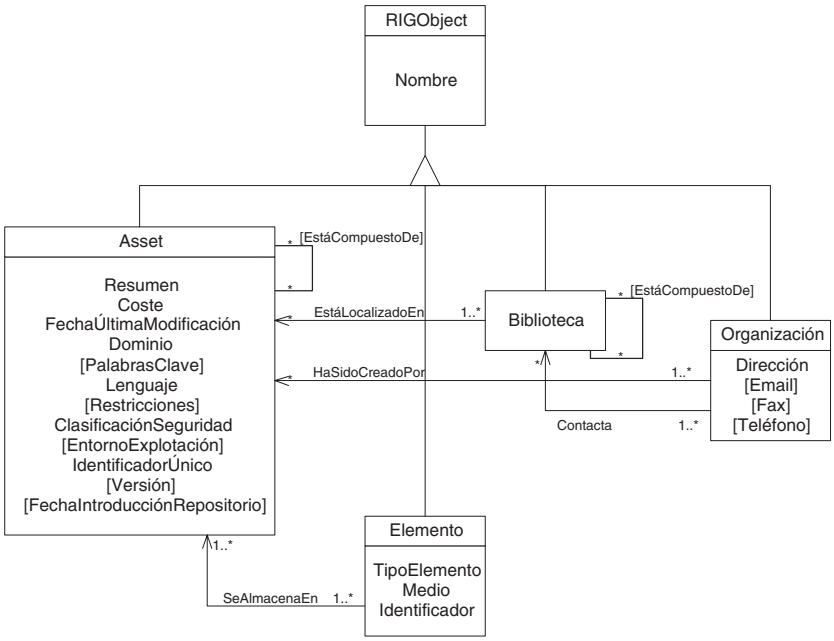

Figura 2.4: Basic Interoperatibility Data Model [NHSE, 1997b].

Del estudio de RIB se derivan una serie de importantes conclusiones sobre el modelo de asset:

- RIB utiliza una extensión del modelo BIDM, estándar de modelo para intercambio de datos propuesto por IEEE.
- Para la definición del modelo de datos subyacente se emplea un modelo de clases, en el que se tienen cuatro clases principales: Asset, Biblioteca, Elemento y Organización. Estas cuatro clases se derivan de una clase raíz denominada RIBObject.
- La definición de los atributos de la clase Asset constituye una referencia válida a seguir.

- La extensión ACF de RIB es una apuesta clara para introducir los aspectos relacionados con la calidad de los assets.

- El punto más débil de esta propuesta es la relación entre assets, contemplando sólo la relación de composición.

Elementos Reutilizables en STARS

El programa **STARS** está patrocinado por la agencia estadounidense **DARPA** (*Defense Advanced Research Projects Agency*), siendo 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*).

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 CFRP se puede ofrecer una base para la adopción y mejora de las prácticas de reutilización dentro de una organización. Aunque el estudio de CFRP se aleja de la temática tratada en el presente apartado, se tiene en [Creps et al., 1992] y en [STARS, 1993] unas fuentes adecuadas para su estudio más detallado.

STARS ALOAF [Solderitsch, 1992], [Solderitsch et al., 1992b] tiene como objetivo la definición de una forma de intercambio de assets entre diferentes bibliotecas de reutilización, y la definición de una plataforma para repositorios sobre la cual construir herramientas portables de reutilización. Para ofrecer una base para establecer una comunicación entre los diversos repositorios orientados a la reutilización de assets, ALOAF incluye **STARS ALF** (*Asset Library Framework*) y **RM** (*Reference Model*) para repositorios.

El modelo de referencia ofrece una visión general de los repositorios y de los elementos que los componen. Establece la terminología básica para repositorios, así como los principales principios en los que se basan éstos. Las características básicas del *Modelo de Referencia de STARS* se muestran en la figura 2.5.

Un repositorio se correspondería con la caja exterior de la figura 2.5, estando compuesto 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 éstos; esta información aparece en la figura 2.5 como *Catálogo de Assets*. Este catálogo de assets contiene la información descriptiva (*Descripciones de los assets*) asociada con los assets y

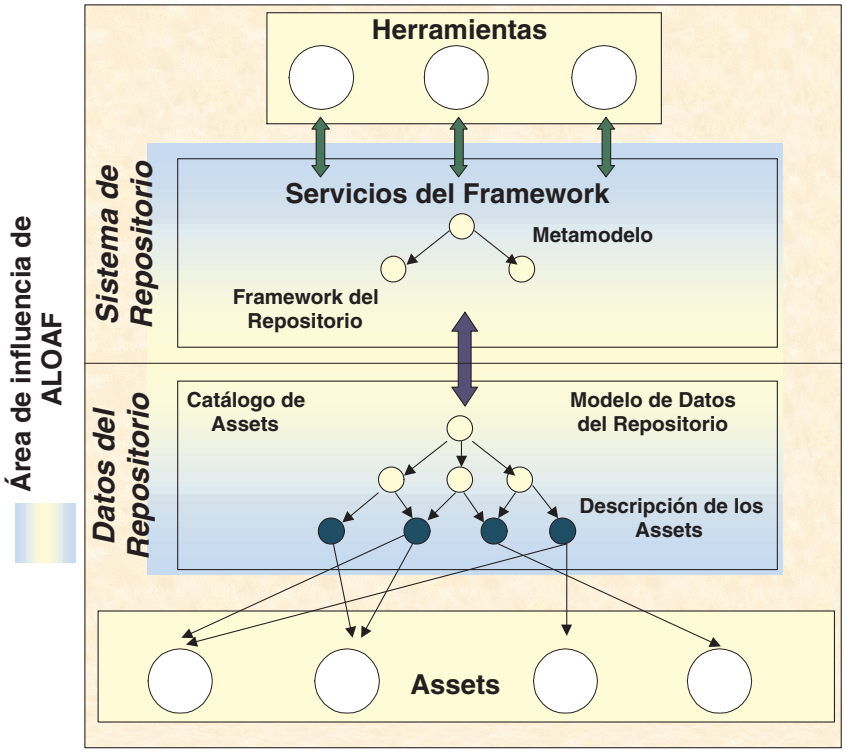

Figura 2.5: Repositorio según el Modelo de Referencia de STARS [Solderitsch et al., 1992b].

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 mismo. 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 a los assets ni a 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.

Uno de los principales objetivos que persigue ALOAF es facilitar el intercambio de assets entre repositorios, para lo cual se 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 serán heredados por el resto de las clases del modelo. El modelo CDM se muestra en la figura 2.6.

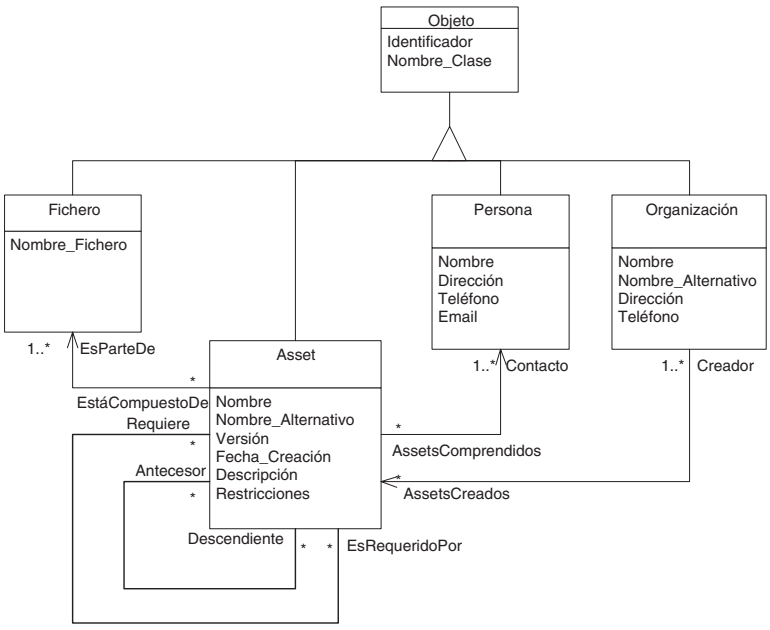

Figura 2.6: El modelo CDM de STARS.

El centro del modelo 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 respectivamente. La clase **Fichero** existe para representar la información sobre los archivos que constituyen los contenidos de un asset. Al utilizar el modelo CDM, los assets se representan como objetos instancias de las clases del modelo.

Según el modelo 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 se encuentran relacionadas a través de las relaciones establecidas en el modelo.

Las características más destacadas de este modelo se pueden resumir en los siguientes aspectos:

- 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. De hecho, el modelo propuesto por el equipo de STARS ALOAF incorpora resultados del RIG como consecuencia de una mutua colaboración[4].
- 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.

RBSE

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

[4]El equipo de ALOAF 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.

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 2.7.

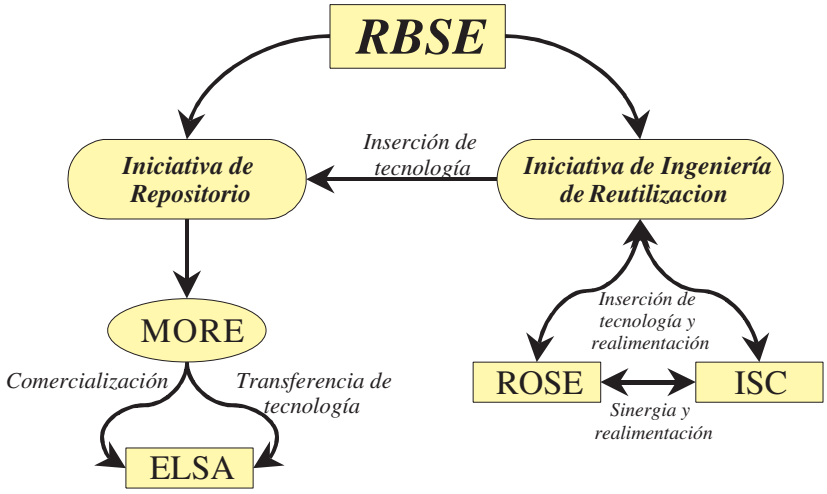

Figura 2.7: Arquitectura de RBSE.

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 URLs en el repositorio de datos.

Para la representación de los assets, ELSA se basa en el estándar propuesto por el RIG a IEEE, esto es, en BIDM.

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, añadiendo éstas los atributos que las definen específicamente [Eichmann et al., 1995].

Las características más relevantes de RBSE en relación con los objetivos de esta tesis se resumen en los siguientes puntos:

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

El Repositorio Microsoft

El repositorio Microsoft [Microsoft, 1997], [Bernstein et al., 1997] presenta una propuesta comercial para la reutilización de componentes[5].

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

[5]Aquí el término componente está haciendo referencia a los elementos software de nivel de implementación que han dado lugar al denominado desarrollo orientado a componentes (*CBD - Component Based Development*).

de componente que sirve como base a la arquitectura de objetos propuesta por Microsoft [Rogerson, 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*) realizado 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 en la 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.

Las principales aportaciones de este repositorio se resumen en los siguientes puntos:

- 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.

- Aunque el Repositorio Microsoft es un repositorio orientado a objetos (*u orientado a componentes*), el motor del mismo se encuentra implementado sobre un motor de bases de datos relacional.

2.2.3 Relaciones entre Elementos de Grano Fino

Un elemento reutilizable de granularidad gruesa con soporte simultáneo de diferentes niveles de abstracción, puede verse como una malla de elementos de grano fino 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 interés de que el elemento reutilizable de grano grueso propuesto soporte simultáneo de diferentes niveles de abstracción, dentro de él aparecen dos familias de relaciones entre sus 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 esta tesis.

De nuevo se va a realizar un repaso por la bibliografía 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.

Relaciones entre Assets Clasificados en el Mismo Nivel de Abstracción

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

En un enfoque sistemático de reutilización del software se puede afirmar que, con independencia del paradigma de desarrollo, en general no sirve de nada la reutilización aislada de un asset. Por contra, éste debe verse acompañado por todo el conjunto de assets que necesita para ofrecer sus servicios.

Esta aseveración es especialmente importante en los desarrollos orientados a objetos, 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, como son el *principio de equivalencia reutilización/revisión*, el *principio de cierre común* y el *principio de reutilización común* [García et al., 1997a].

De esta manera, se está abogando por estructuras complejas de reutilización con independencia del nivel de abstracción en el que se manejen, así se habla de un grano que hace que la reutilización sea efectiva, denominándose de diferentes formas en la bibliografía: **componente** [Stroustrup, 1997], **categoría** [Booch, 1994], **cluster** [Meyer, 1988] o **paquete** [OMG, 1999], [Rumbaugh et al., 1999].

Esta característica se hace patente en los modelos de assets soportados por las diferentes bibliotecas de reutilización estudiadas anteriormente, donde las relaciones entre assets aparecen prácticamente en todos estos modelos, aunque se limitasen a ser simples enlaces entre assets, relegando las connotaciones semánticas al nombre dado a cada tipo de relación soportada por el modelo.

Haciendo un breve repaso por estos modelos, se pueden apreciar las diferentes tendencias sobre las relaciones semánticas estructurales entre assets clasificados en el mismo nivel de abstracción.

El proyecto ITHACA presenta una aproximación por la cual en el SIB se almacenan tanto las descripciones de los objetos software y sus relaciones semánticas. Las relaciones semánticas del modelo ITHACA se clasifican en tres categorías [Constantopoulos and Dörr, 1995]:

- ***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. Están 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 comercial, 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 a continuación:

- 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; aparecen 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 además de las relaciones más típicas.

- El carácter de obligatoriedad con respecto 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.

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[6] presentes en los principales métodos de desarrollo orientados a objetos.

Como referencias representativas se han seleccionado OMT [Rumbaugh et al., 1991], el Método Booch [Booch, 1994], UML 1.x [OMG, 1998], [OMG, 1999], [Booch et al., 1999] y OML [Firesmith et al., 1996]. Del estudio de estos modelos objeto se pueden obtener las siguientes conclusiones:

- Cada propuesta define un modelo objeto diferente, con un núcleo común pero con detalles diferenciales de suma importancia. Estos modelos pueden ser simples, aunque semánticamente ricos, como en el caso de OMT o el Método Booch, o por el contrario pueden ser más complejos como OML o UML.

- Aunque todos estos modelos objetos presentan relaciones semánticas diversas, se puede decir que hay tres relaciones básicas que todos ellos comparten: la relación de asociación, la relación todo/parte y la relación es-un.

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

- Estas tres relaciones centrales deben servir como base para obtener un conjunto de relaciones de semántica más especializada.
- Las relaciones estructurales deben soportar las relaciones semánticas entre elementos del dominio, que en 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].

Como conclusión, se puede afirmar que dos elementos software, pertenecientes al mismo nivel de abstracción, están relacionados entre sí cuando existe un enlace semántico entre ellos.

Relaciones entre Assets Clasificados en Distintos Niveles de Abstracción

Al buscar un elemento reutilizable con soporte simultáneo de varios niveles de abstracción se necesita un mecanismo alternativo que enlace assets clasificados en diferentes niveles de abstracción.

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.

El significado de estas relaciones internivel establece 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.*

Aunque no existen demasiadas referencias sobre relaciones internivel, los trabajos relacionados más destacados se comentan a continuación.

En ITHACA se almacenan assets de diferentes niveles de abstracción, concretamente descripciones de requisitos, descripciones de diseño y descripciones de implementación, lo que 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. El soporte estructural para este soporte multinivel viene representado por los *Application Frames*, bien en su versión genérica (*GAFs*) bien en su versión especializada (*SAFs*).

Otra importante referencia la constituye el **RSRG** (*Reusable Software Research Group*) de la Ohio State University. 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 [Tracz and Edwards, 1989], [Tracz, 1990], [Edwards et al., 1990] 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[7]:

- 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*

[7]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*), de forma que estas relaciones enlazan dichos componentes. 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.

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 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.

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. 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*" [Merriam-Webster Inc., 1996]. En general, se puede definir la reificación como:

1. *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].*

2. *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]. Una aproximación acorde a los intereses del presente

trabajo es la que se ofrece en **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[8] (*o jerarquía de modelos*) donde el metamodelado se basa en la idea de reificar las entidades que forman un cierto tipo de modelo y describir las propiedades comunes del tipo de modelo en forma de un modelo de objetos [Rivas et al., 1997], o el mecanismo clásico para obtener la reflexión.

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

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 una solución software concreta.

[8] 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.

Finalmente, se recogen una serie de conclusiones sobre las relaciones entre assets clasificados en diferente nivel de abstracción:

- 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.
- 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.

Capítulo 3

Modelo de Mecano

La base fundamental del modelo de reutilización que presenta este trabajo está constituida por un elemento reutilizable de grano grueso con soporte simultáneo de varios niveles de abstracción, que recibe el nombre de mecano.

La descripción de la estructura de los mecanos se realiza mediante un modelo orientado a objetos, en el que se recogen todos los aspectos relevantes de estos elementos reutilizables. Este modelo será referenciado a partir de este momento como el **Modelo de Mecano**.

Para la elaboración del modelo se han tomado una serie de decisiones en los apartados relacionados con la definición de una estructura compleja de reutilización, presentados en el *Capítulo 2*.

Este capítulo se organiza de la siguiente manera: en una primera sección se hace una introducción a los mecanos, donde se explica las razones que impulsan a su definición, así como los *requisitos* principales que ha de cumplir esta estructura de reutilización. En la sección dos de este capítulo se presentan los elementos que conforman un mecano, siguiendo un enfoque constructivo en el que se detallan los aspectos concernientes a *los niveles de abstracción soportados, la estructura de sus componentes de grano fino o assets y las relaciones entre éstos*, esto es, se definen las bases de la estructura del modelo de componente reutilizable. En la sección tres se establece el Modelo de Mecano, diferenciándose por una parte lo que es el modelo de componente reutilizable y por otro el modelo de repositorio capaz de soportarlo. Ambos modelos están sumamente relacionados y se utilizan técnicas semi-formales (UML) para ese cometido. Por último, la cuarta sección presenta la definición formal del modelo de componente reutilizable, soportada por la teoría de grafos.

3.1 Introducción a los Mecanos

La propuesta realizada para la participación en el proyecto coordinado **MENHIR**[1], desemboca en diversas experiencias prácticas en el campo de la reutilización del software, entre las que cabe destacar [Villa, 1997] y [Martínez and Maudes, 1997], donde se pone de manifiesto la necesidad de establecer un modelo de reutilización que mejore el proceso de reutilización en ambientes de desarrollo reales.

Determinar la tipología de los elementos software, núcleo del modelo de reutilización definido, es un aspecto fundamental que se traduce en una de las claves principales para aumentar la potencia del proceso de reutilización a través de un conjunto de principios básicos que se recogen en [García et al., 1998a]:

- **Aumento del nivel de abstracción de la reutilización:** Busca que el alcance de la reutilización del software se dirija hacia niveles de mayor abstracción que la implementación.

- **Soporte de diferentes niveles de abstracción de forma simultánea:** Los elementos reutilizables de grano grueso propuestos están compuestos por un conjunto de assets, o elementos de grano fino, interrelacionados entre sí. Sin embargo, y en relación con el aumento del nivel de abstracción del proceso de reutilización, se hace especial hincapié en que haya assets representantes de los diferentes niveles de abstracción que se distinguen en todo proceso de desarrollo del software.

- **Soporte para la trazabilidad:** Dirige la navegación por los diferentes assets que forman un mecano a través de la red de enlaces entre ellos, con especial interés por los enlaces entre los assets clasificados en diferentes niveles de abstracción, que son los que mayor potencia dan al proceso de reutilización.

- **Integración dentro del proceso de reutilización:** Esto es, el elemento reutilizable debe estar presente en todas las actividades propias tanto del *desarrollo para reutilización* como del *desarrollo con reutilización.*

Los mecanos son elementos reutilizables que se ajustan a las premisas expuestas, pudiéndose definir como:

[1] CICYT-TIC97-0593-C05-01

Definición 3.1 MECANO:

Se denomina mecano a un conjunto de elementos software reutilizables de grano fino, clasificados en diferentes niveles de abstracción 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.

3.1.1 Requisitos de los Mecanos

Como paso previo a la exposición detallada de los apartados que se han tenido en consideración para la definición del Modelo de Mecano, se van a enumerar y comentar aquellos requisitos o restricciones que se impusieron para la definición de dicho modelo o estructura. Algunos de estos requisitos se obtienen directamente de la definición de mecano, sin embargo otros se derivan del estudio detallado del proceso general de reutilización, así como de su puesta en marcha en experiencias reales.

Estos requisitos o restricciones son:

- *Cada uno de los assets componentes de un mecano debe estar clasificado en un determinado nivel de abstracción.* La fase del ciclo de vida del asset va a marcar el nivel de abstracción en el que se clasifica, distinguiéndose tres niveles de abstracción: requisitos, diseño e implementación.

- *Existencia de dos tipos de relaciones entre assets: relaciones intranivel y relaciones internivel.* Dado 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 clasificados en un mismo nivel de abstracción y las relaciones entre assets clasificados en 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 [Parnas et al., 1989], [Krueger, 1992], [Cybulski, 1997].

  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

asumen que la reutilización sistemática pueda afectar a varios niveles de abstracción al mismo tiempo. Para solucionar este problema, los mecanos ofrecen una perspectiva multinivel de abstracción, de forma que siempre representan algún tipo de refinamiento de un elemento software a través del tiempo.

Este mismo requisito lleva a la situación de que cualquier mecano debe contar siempre con al menos una relación internivel.

- *Todo asset debe ser de un tipo predefinido, que indique de qué clase de elemento software se trata.* Cada uno de los elementos de grano fino componentes de un mecano debe pertenecer a un tipo de asset predefinido, el cual recoja las características propias de ese tipo de elemento software y permita establecer un control sobre con que otros assets puede relacionarse.

- *Debe haber 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 que dan lugar a nuevos tipos de elementos software susceptibles de ser reutilizados.

- *La estructura de los elementos de grano fino (assets), que componen un mecano, debe dar soporte a su información lógica y física, además de ofrecer detalles sobre la seguridad, la calidad y los aspectos administrativos de los mismos.* 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 mecanos.* 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 entre los assets que forman un mecano deben quedar semánticamente especificados.* 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 construcción de los mecanos.* Restricciones 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 ligado a uno o a varios dominios.* La reutilización vertical del software en dominios de aplicación concretos es la forma más efectiva de obtener los mayores beneficios de la reutilización. La perspectiva de grano grueso, unida a su capacidad multinivel de abstracción, convierte a los mecanos en unos elementos reutilizables que se ajustan muy bien al desarrollo de software vertical utilizando reutilización.

- *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.

3.2 Elementos de un Mecano

Teniendo en cuenta la definición de lo qué es un mecano, así como los principios básicos que debe conformar y los requisitos que debe satisfacer cada uno de estos elementos reutilizables de grano grueso, se define el modelo de componente reutilizable que especifica su estructura.

Para la mejor comprensión del modelo, se explica cada uno de los apartados que tenidos en cuenta para su definición, siguiendo un razonamiento paralelo al realizado en el *Capítulo 2* al presentar el estado del arte de los elementos a tener en cuenta para definir una estructura compleja de reutilización.

3.2.1 Niveles de Abstracción Soportados por 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 adopta el criterio más extendido en la bibliografía [Constantopoulos et al., 1992], [Bellinzona et al., 1993], [Karlsson, 1995], [McClure, 1997] 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 al menos otro criterio de clasificación basado en algún otro factor para la selección y recuperación de los assets componentes, como podría ser un sistema de clasificación basado en facetas al estilo del propuesto por Rubén Prieto-Díaz [Prieto-Díaz, 1989], [Prieto-Díaz, 1991] (*aunque este aspecto puede verse como un servicio de la biblioteca de reutilización, ortogonal a la definición de los mecanos*).

Así, los mecanos pueden estar compuestos por un máximo de tres niveles de abstracción (*y un mínimo de dos*), que en orden decreciente de abstracción son **nivel de requisitos, nivel de diseño** y **nivel de implementación**.

Esto conduce a contar con cuatro tipos de mecanos según su configuración de niveles de abstracción:

1. **R-D-I:** Es el caso más completo, y al cual se debe tender para obtener los mayores beneficios de un modelo de reutilización sistemática. Se tienen assets clasificados en los tres niveles de abstracción (*requisitos, diseño e implementación*) y relaciones internivel entre ellos.

2. **R-D:** Representa casos en los que se tienen especificaciones y diseños relacionados por relaciones internivel. Puede ser un caso típico de definición del asset base de una determinada línea de producto.

3. **R-I:** Se tienen assets en los niveles de especificación e implementación, pero no en el nivel de diseño. Este tipo de mecanos puede darse en entornos de desarrollo en los que, partiendo de especificaciones de requisitos, se genere automáticamente el código fuente.

4. **D-I:** Se tienen assets en todos los niveles de abstracción a excepción de en el nivel

de requisitos. Puede ser una configuración típica donde se quiera establecer un repositorio de mecanos que representen componentes de utilidad o bibliotecas de tipos abstractos de datos junto con posibles implementaciones.

3.2.2 Modelo de los Assets Componentes de un Mecano

El modelo propuesto para los elementos reutilizables de grano fino (*assets*) que componen un mecano está basada en el estándar IEEE para intercambio de información entre bibliotecas de reutilización, o modelo **BIDM** [IEEE, 1995].

El centro de este modelo, que se muestra en la figura 3.1, es la clase **Asset**, pero existen otras entidades con las que se relaciona; estas entidades están representadas por las clases **TipoAsset, Representación** y **Creador**.

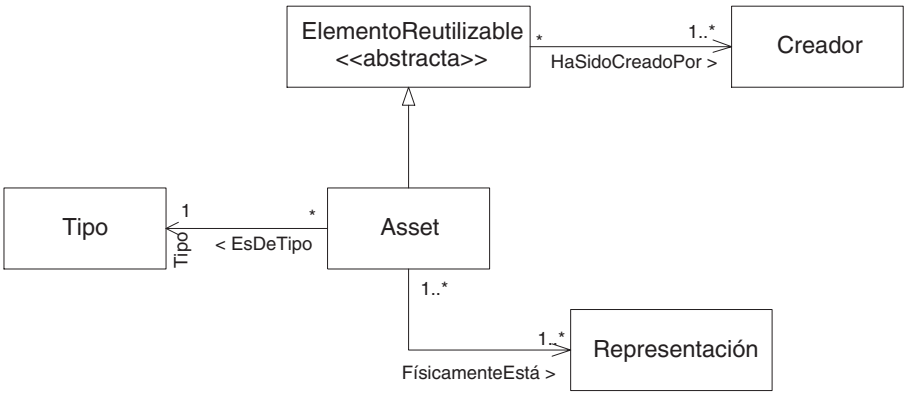

Figura 3.1: Modelo básico del asset componente de un mecano.

Aparece en el modelo de la figura 3.1 la clase denominada **ElementoReutilizable** como superclase de **Asset**; esto es así, porque en el modelo de reutilización que se presenta se manejan dos tipos de elementos reutilizables: uno de grano fino, *asset*, y otro de grano grueso, *mecano*, compartiendo ambos un conjunto de atributos de información que se recogen en la clase **ElementoReutilizable**.

Clase ElementoReutilizable

Con esta clase abstracta se recoge los atributos que son comunes a los elementos reutilizables manejados, con independencia de su granularidad. Estos son:

- **Identificador:** Un campo de identificación único en el sistema.

- **Nombre:** Nombre o título del elemento reutilizable.
- **Resumen:** Descripción textual con la explicación o definición del elemento reutilizable.
- **FechaCreación:** Fecha en la que el elemento reutilizable fue creado.
- **FechaModificación:** Fecha de última modificación del elemento reutilizable.
- **PalabrasClave:** Lista de palabras o frases que describen los temas relacionados con el elemento reutilizable.
- **Entorno:** Entorno de destino para el elemento reutilizable (*plataforma hardware, sistema operativo...*).
- **Restricciones:** Información legal sobre el uso del elemento reutilizable, incluyendo *copyright, patentes, derechos de explotación, limitaciones de exportación* y *licencias*.
- **NivelSeguridad:** Indica el nivel de seguridad mímimo con el que se debe contar para acceder al elemento reutilizable. Será el valor máximo de entre el valor de seguridad del elemento reutilizable y los valores de seguridad de todos sus componentes.
- **Coste:** Cuantía que debe pagarse por la reutilización del mismo.
- **Versión:** Designación de la versión de un elemento reutilizable.
- **Certificación:** Nivel de certificación del elemento reutilizable. Es otorgado por el administrador de la biblioteca de reutilización. Se establecen los siguientes niveles: **desconocido** (*el elemento reutilizable no ha sido revisado*), **revisado** (*el administrador ha comprobado que se encuentra completo*), **auditado** (*el elemento reutilizable ha pasado por un proceso de auditoría, fijado por el modelo de reutilización en curso, que garantice su corrección y compleción [Manso et al., 1999b]*) y **cualificado** (*el elemento reutilizable ha pasado por un proceso de cualificación que incluye, además de una auditoría, la aplicación de las métricas que el modelo de reutilización establezca [Manso et al., 1998], [Sáez et al., 1999]*).
- **NúmeroExtracciones:** Número de veces que el elemento reutilizable ha sido reutilizado.
- **Comentarios:** Información que los desarrolladores con reutilización introducen sobre la experiencia de reutilización práctica del elemento reutilizable.

Clase Asset

La clase **Asset** representa un asset lógico genérico, esto significa que, con sus atributos y los heredados de la clase **ElementoReutilizable**, expresa la definición de un asset, pero sin reflejar ninguna connotación física del mismo.

Esta clase presenta los siguientes atributos:

- **FechaIntroducción:** Fecha en que el asset fue introducido en el repositorio.
- **NivelAbstracción:** Indica la fase del ciclo de vida en que fue generado el asset. Para ser consecuentes con la faceta del nivel de abstracción, este atributo debe tomar uno de los valores siguientes: **requisitos, diseño** o **implementación**. El valor de este atributo vendrá marcado por los niveles de abstracción que sean admisibles para el tipo de asset al que pertenece.
- **Metodología:** Metodología o método de desarrollo seguido para la generación del asset.
- **Idioma:** Lengua en la que se encuentra documentado el asset.

Clase TipoAsset

Como se indicaba en los requisitos que debía cumplir todo mecano, cada uno de los assets componentes de un mecano debe pertenecer a un tipo de asset, el cual viene representado por la clase **TipoAsset**. Esta entidad tiene como cometido caracterizar al asset, así como introducir un control en la forma en que los assets se relacionan.

Los atributos con los que cuenta esta clase son:

- **Identificador:** Campo de identificación único en el sistema.
- **Nombre:** Nombre del tipo.
- **NivelesAbstracciónAdmitidos:** Lista de los niveles de abstracción admitidos por el tipo.
- **Paradigma:** Paradigma de desarrollo bajo el que se ha generado. En la mayoría de los casos el paradigma será *Orientado a Objetos* o *Estructurado*.

La clase **Asset** se relaciona con la clase **TipoAsset** a través de la asociación **EsDeTipo**, que refleja la obligatoriedad de que todo asset tenga un tipo.

Clase Representación

Determina la parte física de un asset, es decir, toda la información relacionada con la localización física del asset.

Los atributos que presenta la clase **Representación** son los siguientes:

- **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.
- **Herramienta:** Producto software que se ha utilizado para crear el asset (*herramienta CASE, procesador de textos, lenguaje de programación...*), indicando versión de la misma.
- **Formato:** Formato del fichero (*ASCII, LaTeX, PDF, MS Word, postscript, binario...*).
- **Tamaño:** Tamaño del asset (*KB, páginas...*).
- **Medio:** Medio por el que puede obtenerse el asset (*CD-ROM, papel, vídeo...*).
- **Comentarios:** Posibles notas sobre el formato físico del asset.

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.

Clase Creador

Esta clase representa a la organización y/o al autor(es) responsable(s) de la creación y mantenimiento de los elementos reutilizables.

Los atributos de la clase **Creador** son:

- **Identificador:** Es un identificador único dentro del sistema.

- **Nombre:** Es el nombre del responsable.
- **RazónSocial:** Razón social de la empresa.
- **IdentificadorFiscal:** NIF o CIF del responsable.
- **Dirección:** Dirección postal.
- **Email:** Correo electrónico.
- **Teléfono:** Lista de números de teléfono de contacto.
- **Fax:** Número de fax.
- **Comentarios:** Comentarios de relevancia sobre el creador del asset.

La clase **ElementoReutilizable** está relacionada con la clase **Creador** mediante la asociación **HaSidoCreadoPor**, de forma que todo elemento reutilizable tenga asociado al menos un responsable, ya sea una organización o una persona concreta.

Añadiendo los atributos identificados en cada una de las clases del modelo presentado en la figura 3.1, se refina y se completa en el modelo de clases (*expresado a nivel conceptual*) tal y como se muestra en la figura 3.2.

3.2.3 Relaciones entre los Assets Componentes de un Mecano

Las relaciones entre assets tienen un papel protagonista dentro del entorno de reutilización sistemática propuesto, ya que son la base de importantes actividades con los mecanos: *navegación a través de sus componentes, trazabilidad, procesos de recuperación, versionado...*

Al hablar de relaciones entre assets se hace referencia a las relaciones semánticas que se dan entre los objetos del universo de discurso[2] 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[3] que aporte un conjunto

[2]El universo de discurso hace referencia en este trabajo al desarrollo del software.

[3]Básicamente el modelo objeto de referencia va a ser el de UML 1.x ampliado con la relación de reificación.

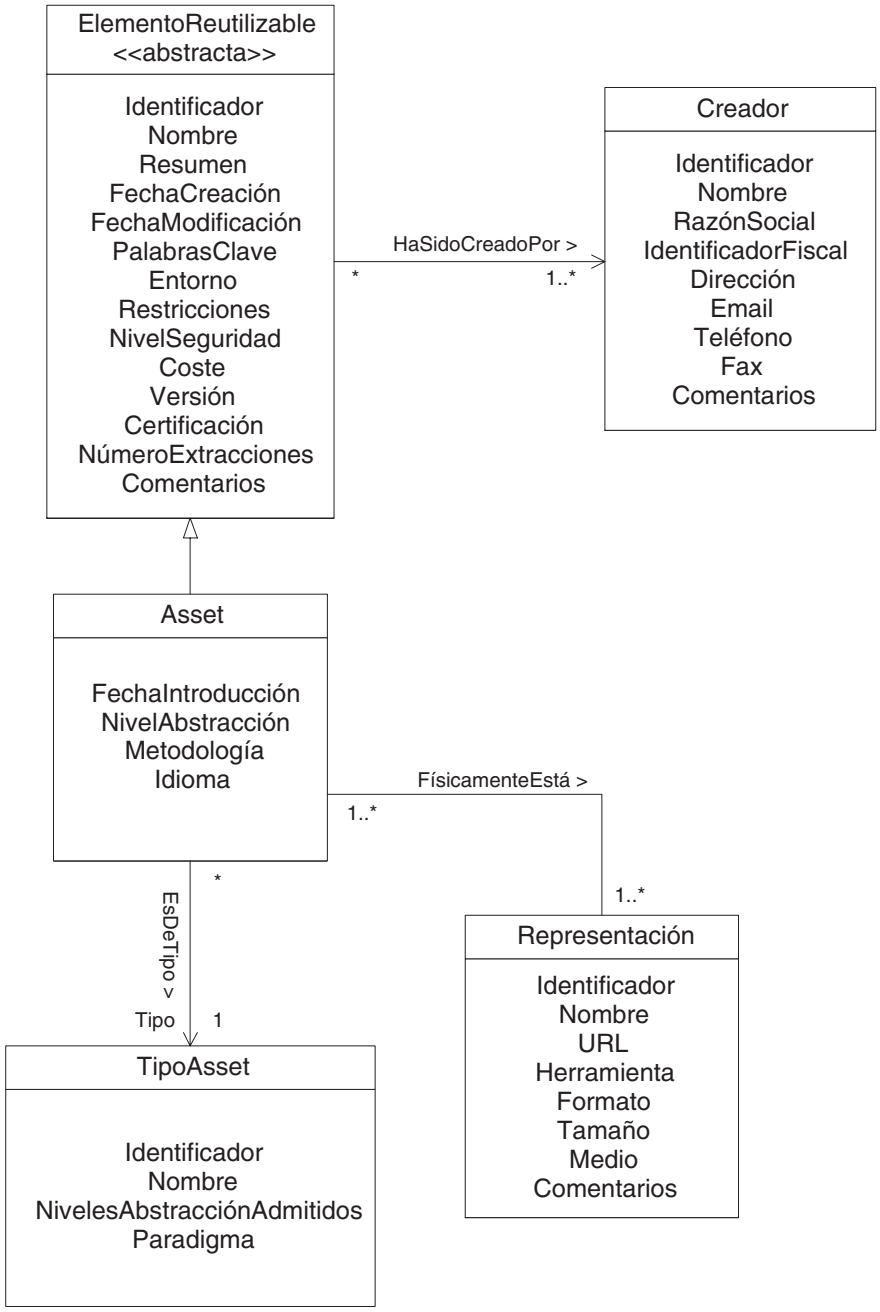

Figura 3.2: Modelo refinado del asset componente de un mecano.

reducido de relaciones estructurales, que sirvan para caracterizar los tipos de relaciones semánticas entre los assets [García et al., 1998b].

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, pudiéndose apreciar en la figura 3.3.

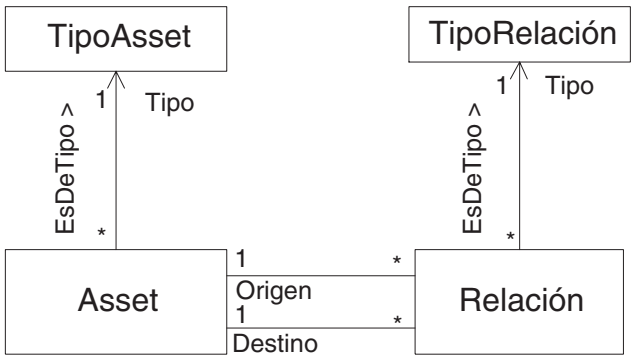

Figura 3.3: Relaciones entre los assets componentes de un mecano.

- Pero a su vez cada uno de los tipos de relación semántica está soportado por una relación estructural definida en el modelo objeto de referencia, y que sirven para caracterizar los tipos de relaciones semánticas. Este es el tercer nivel de abstracción, correspondiéndose con un nivel meta.

En la figura 3.4 se muestra un diagrama de objetos que representa 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[4]. 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 **Extensión** en el ejemplo utilizado[5].

[4]Los assets **A1, A2** y **A3** tienen un tipo de asset (*ADT, Clase, Clase respectivamente*).

[5]Para representar de una forma gráfica la relación estructural que caracteriza al tipo de relación semán-

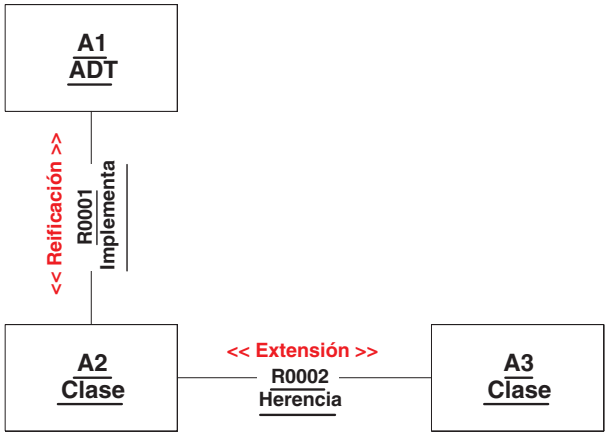

Figura 3.4: Diagrama de objetos que representa un mecano.

El ejemplo que ilustra la figura 3.4 se utiliza en la figura 3.5 para enfatizar en los diferentes niveles de abstracción que se tienen en la definición de las relaciones entre elementos de grano fino dentro del Modelo de Mecano. El *nivel de modelo* presente en la figura 3.5 se corresponde con la parte central del modelado de las relaciones entre assets dentro del Modelo de Mecano, tal y como se mostraba en la figura 3.3.

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 para la construcción de los mecanos.

Definición de las Relaciones Estructurales en el Modelo de Mecano

Todas las relaciones semánticas que se puedan establecer entre los diferentes tipos de assets están caracterizadas por un conjunto reducido de relaciones estructurales que se encuentran recogidas en el modelo objeto de referencia.

En consecuencia, las relaciones estructurales en el Modelo de Mecano determinan las familias de enlaces semánticos permitidos entre las entidades del problema.

Uno de los aspectos más relevantes de los mecanos como estructura compleja de reutilización es que sus componentes (*assets*) van a estar clasificados en diferentes niveles de

tica, se ha optado por recurrir a la notación de asociación de UML 1.x junto con un *estereotipo* que refleja el tipo de relación estructural.

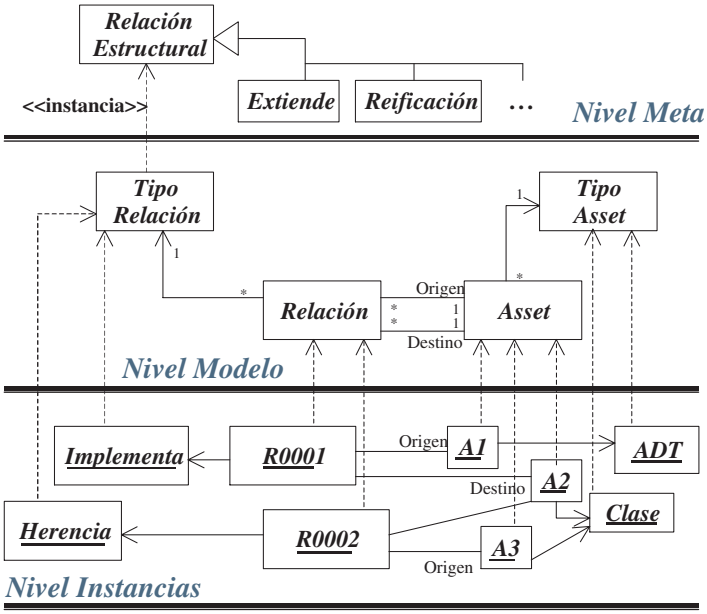

Figura 3.5: Niveles de abstracción en la definición de los mecanos.

abstracción[6], especialmente cuando se hace 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:

[6]El nivel de abstracción de un mecano será el nivel de abstracción más alto de sus componentes.

Definición 3.2 REIFICACIÓN:

Dados X e Y dos assets pertenecientes a diferentes niveles de abstracción, se dice que el asset Y reifica al asset X si y sólo si Y es un refinamiento del asset X.

Con la reificación se cubre el apartado 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 (*ser parte de*) 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, todos ellos clasificados en el mismo nivel de abstracción. 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:

Definición 3.3 AGREGACIÓN:

Dados X e Y dos assets pertenecientes al mismo nivel de abstracción, se dice que el asset Y está relacionado con el asset X mediante una relación de agregación si y sólo si el asset Y forma parte de la estructura de X, y además puede reutilizarse con independencia de la reutilización de X.

Definición 3.4 COMPOSICIÓN:

Dados X e Y dos assets pertenecientes al mismo nivel de abstracción, se dice que el asset Y está relacionado con el asset X mediante una relación de composición si y sólo si 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.

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:

Definición 3.5 USA A:

Dados X e Y dos assets pertenecientes al mismo nivel de abstracción, se dice que el asset Y usa al asset X si y sólo si el asset Y depende del comportamiento especificado por el asset X para su definición o para completar su comportamiento.

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:

Definición 3.6 EXTENSIÓN:

Dados X e Y dos assets pertenecientes al mismo nivel de abstracción, se dice que el asset Y extiende al asset X si y sólo si el asset Y está definido en función de la definición del asset X.

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:

Definición 3.7 ASOCIACIÓN:

Dados X e Y dos assets pertenecientes al mismo nivel de abstracción, se dice que el asset X está asociado al asset Y si y sólo si existe un enlace semántico bidireccional entre ambos assets.

En resumen se puede decir que el modelo objeto subyacente debe proporcionar un el conjunto de relaciones estructurales anteriormente descritas. Dicho modelo objeto puede derivarse del propuesto en UML 1.x [OMG, 1999] con la inclusión de la relación de reificación.

Caracterización de las Relaciones Estructurales

Una vez identificadas y definidas las relaciones estructurales necesarias en el espacio de reutilización de definición de los mecanos, se van a presentar un conjunto de propiedades que las caracterizan. 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.**

  El orden de una relación viene definido por el número de roles que intervienen en cada enlace [Blaha and Premerlani, 1998].

  Para el modelado de las relaciones entre los elementos de grano fino constituyentes de un mecano se ha optado por la utilización de relaciones binarias, como se puede deducir de las definiciones anteriormente enunciadas.

  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 cómo 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 condición de reutilizar también todos aquellos assets relacionados con el primero.
  - ◇ 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.
  - ◇ 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, 1996a], [Girow, 1996b] 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.**

  Se entiende por dirección de una relación el enlace semántico dirigido entre un origen y un destino.

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 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 tiene una polaridad semántica que se recoge en la tabla 3.1, a excepción de la asociación, considerada bidireccional por definición por diversos autores [Booch, 1994], [Rumbaugh et al., 1999].

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"*
Usa a	*Del asset "cliente" al asset "servidor"*
Extensión	*Del asset "derivado" al asset "base"*

Tabla 3.1. Relaciones estructurales con una polaridad semántica definida.

- **Carácter de las relaciones.**

  Se entiende por carácter de una relación estructural a la característica que cualifica un enlace semántico dirigido entre un asset origen y un asset destino con respecto al proceso de reutilización.

  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, permite dotar al modelo de reutilización de una notable flexibilidad, dado que no sólo permite crear elementos complejos en desarrollos para reutilización para posteriormente recuperarlos tal cual, sino que podrían ser compuestos automáticamente en tiempo de reutilización, aprovechando la información aportada por la red de enlaces semánticos, conforme a unas necesidades específicas de los desarrolladores con 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; así pues, 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, haciendo especial hincapié en su relación con un proceso de construcción de mecano desde un prisma de generación basada en composición.

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, desde el punto de vista del enfoque sistemático que se ofrece en este trabajo, 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 favoreciendo el enfoque generativo por composición dentro del proceso de reutilización, gracias a que 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.

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.

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 dos casos extremos:

- ◇ Dos enlaces débiles en ambas direcciones de navegación, lo cual permite introducir una serie de tipos de relaciones semánticas muy importantes en el espacio de reutilización que, aunque no tendrían una implicación directa en el proceso de construcción generativa de mecanos, son imprescindibles en la definición de elementos software reutilizables complejos, como por ejemplo el caso de la relación de versionado.
- ◇ Dos enlaces fuertes en ambas direcciones de navegación. Esto permite expresar situaciones en las que dos tipos de assets se necesitan mutuamente, pero no existe una relación "*todo/parte*" entre ellos. Inicialmente, en los primeros estudios realizados sobre el carácter de las relaciones semánticas [García et al., 1998b], esta posibilidad no fue contemplada. Pero al llevar los trabajos teóricos al terreno práctico se vieron sus ventajas, al sustituir aquellos casos donde se debían plantear dos relaciones "**usa a**" dirigidas en sentidos contrarios entre dos assets.

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**	Reutilizarel 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
	Fuerte	**Fuerte**	**Documenta**	**Caso de Uso**	**Plantilla de Descripción de Requisitos Funcionales**	Reutilizar el Caso de Uso implica reutilizar su descripción funcional y viceversa.

Tabla 3.2. Carácter de las relaciones estructurales frente al proceso de reutilización.

La tabla 3.2 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 enfoque de construcción de mecanos mediante generación basada en composición, 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.

Relaciones en el Modelo de Mecano

Una vez que se ha quedado explicado 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 3.3 aparece el núcleo de la parte del Modelo de Mecano referente a las relaciones entre los assets componentes, pero es en la 3.6 donde esta parte queda reflejada en su totalidad.

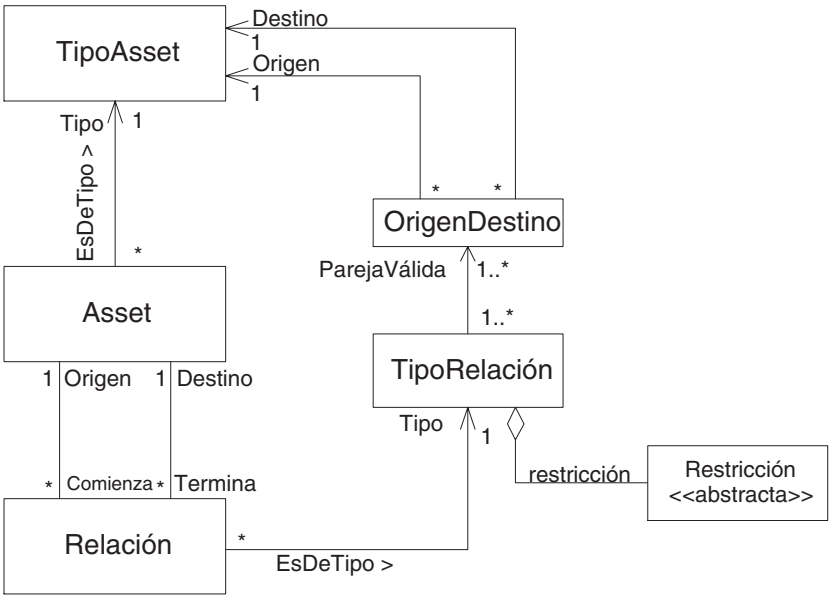

Figura 3.6: Modelado de las relaciones en el Modelo de Mecano.

La clase **Relación** es la responsable de establecer los enlaces entre los assets, es decir, es la encargada 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 cuanto a las asociaciones de la clase Relación se tiene que[7]:

[7]Para expresar las restricciones dentro del Modelo de Mecano se va a utilizar **OCL** (*Object Constraint Specification*) [OMG, 1999], [Warmer and Kleppe, 1999].

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** tiene los siguientes atributos:

- **Nombre:** Nombre único del tipo de relación semántica.
- **Estructura:** Tipo enumerado que indica cuál es la relación estructural que le sirve como definición, pudiendo tomar uno de los valores siguientes: *Reificación, Agregación, Composición, Usa A, Asociación Fuerte* y *Asociación Débil.*
- **EsInternivel:** Bandera que indica si el tipo de relación semántica es o no internivel.
- **CarácterOrigenDestino:** Carácter con respecto al proceso de reutilización del enlace dirigido del origen al destino del mismo.
- **CarácterDestinoOrigen:** Carácter con respecto al proceso de reutilización del enlace dirigido del destino al origen del mismo.

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 qué 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 -> exists (_origen: Asset, _destino: Asset,
                               _orgdes: OrigenDestino |
                               _orgdes.Origen.Nombre= _origen.Tipo.Nombreand
                               _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 **Restricción** 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 las relaciones de tipo 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 3.7 se presenta de nuevo la parte del Modelo de Mecano referente a las relaciones entre sus assets componentes, pero contemplando los atributos y métodos identificados.

3.3 Modelo de Mecano - Uso de Técnicas Semi-Formales (UML)

Tras la exposición de los elementos que forman la estructura compleja de reutilización denominada mecano, se va a proceder en los siguientes subapartados a la integración de estas piezas en un único modelo, el Modelo de Mecano. Este modelo equivale al modelo de repositorio, donde aparece inmerso el modelo de componente reutilizable, el mecano, junto con toda la información necesaria para su clasificación, cualificación, recuperación, mantenimiento y uso .

Siguiendo con la dinámica de este capítulo, se recurre a UML 1.x como técnica de modelado semi-formal, para plasmar los diagramas oportunos.

3.3.1 La Entidad Mecano

En la sección anterior se ha desgranado las entidades que forman un mecano. Ahora le llega el turno al estudio de la entidad que da nombre al modelo: el mecano.

Como se ha expuesto a lo largo de este capítulo, un mecano es un elemento reutilizable, de grano grueso, compuesto por assets. Pero además, y pensando en la manera de organizar

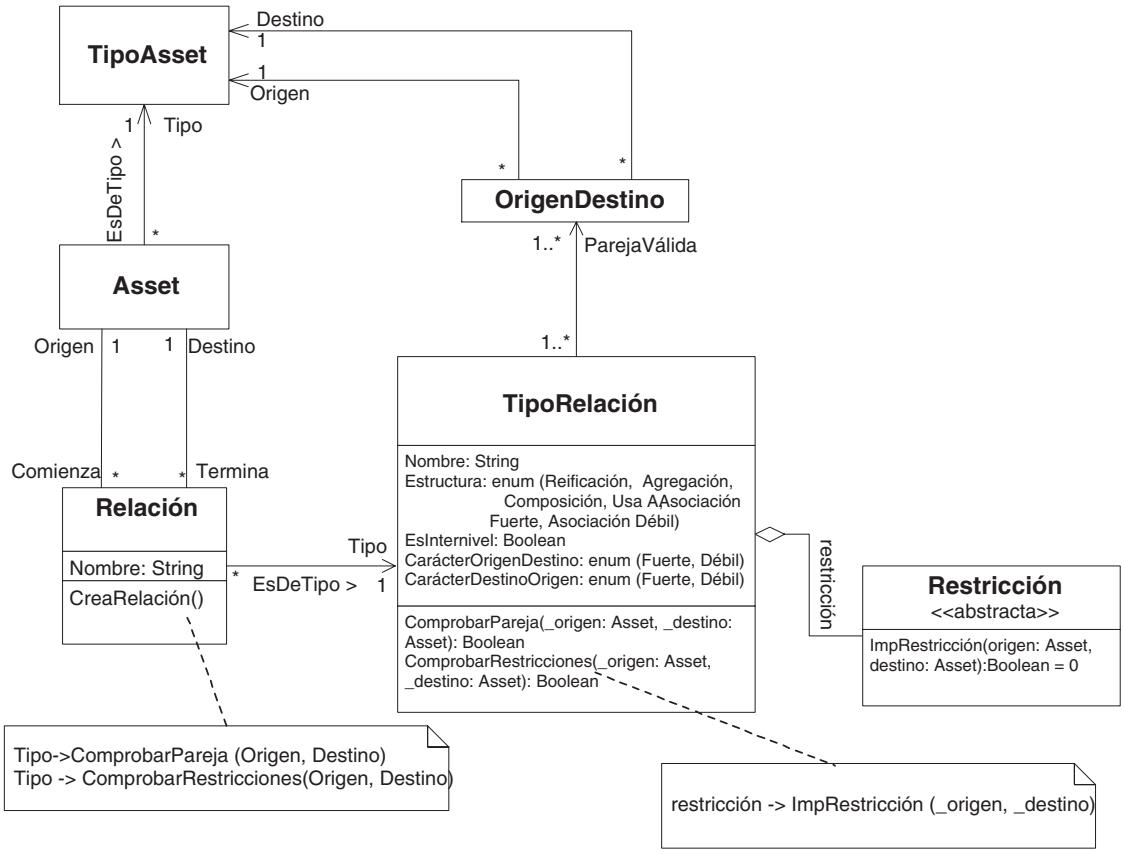

Figura 3.7: Modelo de las relaciones en un mecano con un mayor grado de detalle.

los mecanos dentro del modelo de reutilización propuesto, se tiene que todo mecano está en un dominio de aplicación, que puede representar productos de una o varias líneas de producto dentro de dicho dominio. Con estas bases, se puede elaborar un primer modelo preliminar que se presenta en la figura 3.8.

La clase **Mecano** representa a la entidad mecano dentro del Modelo de Mecano. Hereda todos los atributos de la clase **ElementoReutilizable** y añade el atributo **Paradigma**; atributo que representa el paradigma bajo el que ha sido desarrollado o creado el elemento reutilizable. En el caso de los mecanos el paradigma será *Orientado a Objetos*, *Estructurado* o *Mixto*.

Que un mecano se modele como una agregación y no una composición se debe a que 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

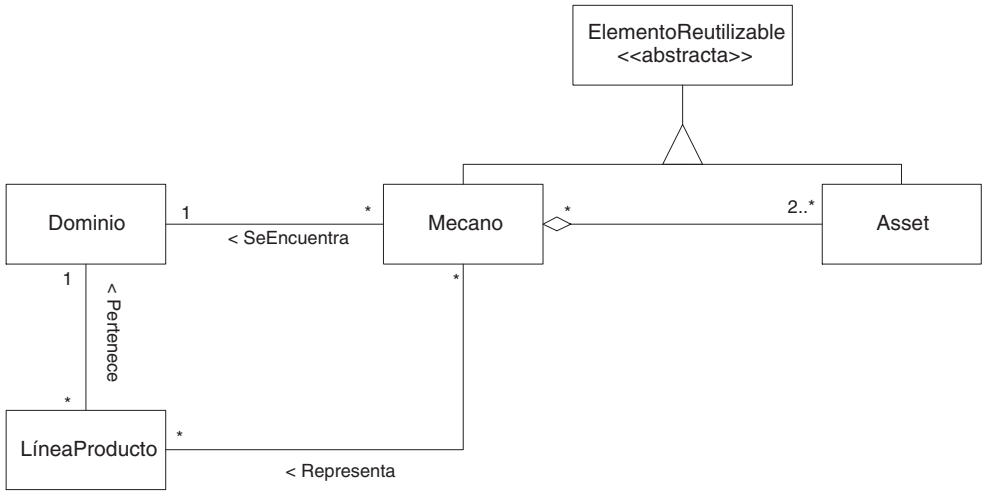

Figura 3.8: Modelo preliminar de un mecano.

eliminación de sus componentes[8].

Además, los mecanos están clasificados en un dominio de aplicación, que representa el área de aplicación, actividad o conocimiento del mecano. Este concepto viene definido por la clase **Dominio** y la vinculación de un mecano a un dominio se presenta mediante la asociación **SeEncuentra**.

Como un dominio puede ser un concepto demasiado amplio, se ha introduce el concepto de línea de producto, como una manera de organizar el dominio en subdominios. Así, un mecano puede representar productos o definiciones de una o varias líneas de productos del dominio en el que está clasificado. Esto se representa en el modelo mediante la clase **LíneaProducto** y las asociaciones **Pertenece** (*que relaciona la línea de producto con un dominio*) y **Representa** (*que relaciona al mecano con las líneas de producto*).

Mecano

```
self.LíneaProducto->forAll  (LíneaProductolp | lp.Dominio=self.Dominio)
```

En cuanto a las clases **Dominio** y **LíneaProducto** tienen los típicos atributos de identificación y de descripción (*figura 3.9*).

[8]La connotación semántica de la relación de agregación entre el mecano y los assets se corresponde con lo que UML 1.x 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.

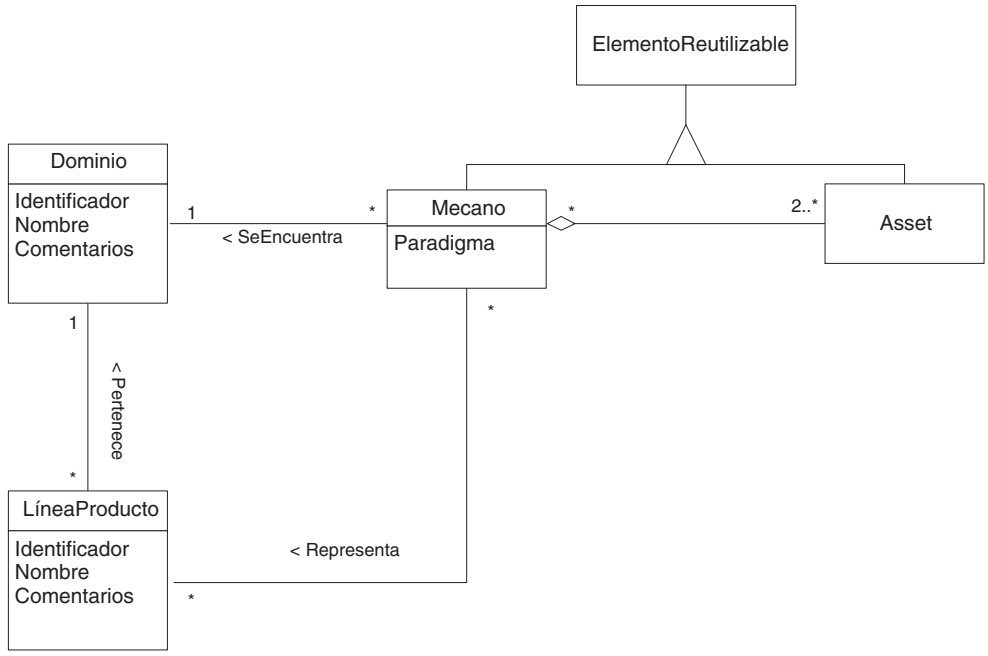

Figura 3.9: Modelo preliminar refinado de un mecano.

3.3.2 Alternativas en los Mecanos

Un problema no trivial es el que surge del carácter evolutivo intrínseco a los elementos software. Se tiene que los elementos componentes de un mecano van a evolucionar tanto por actividades propias de mantenimiento, como por el propio proceso de reutilización que realimenta el repositorio con adaptaciones realizadas. A este problema se une la complicación adicional de que los mecanos, al ser elementos de grano grueso, pueden verse como configuraciones de assets que evolucionan junto con sus relaciones.

Para evitar caer en la explosión combinacional, que estos casos provocan, intentando no perder flexibilidad en cuanto al proceso de reutilización, se han introducido las siguientes restricciones en el Modelo del Mecano:

1. Se dice que se tiene una situación *ambigüedad* cuando un asset se relaciona con otros assets mediante el mismo tipo de relación semántica, y no se está en situación de determinar si esas relaciones son todas necesarias o son alternativas.

2. Un mecano almacenado en el repositorio debe estar preparado para ser reutilizado tal cual ha sido creado, de forma que no puede contener ambigüedades, lo cual implica que no puede haber alternativas dentro de un mecano.

3. Cuando se producen alternativas que se quieren contemplar en el repositorio, se originan nuevos mecanos que representan diferentes configuraciones de un mismo producto, estando estos enlazados por relaciones tipo ***ancestro/derivado*** (*situación que se refleja en la figura 3.10*).

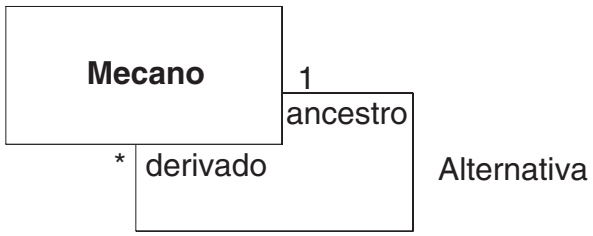

Figura 3.10: Alternativas en los mecanos.

4. Se denomina versión de un asset, a aquel asset que puede sustituir al primero sin modificar la topología de enlaces del mecano al que pertenece. Estos assets estarán relacionados mediante enlaces que conformen al tipo de relación semántica que soporte el versionado.

5. El mecano siempre tiene constancia de las versiones activas de sus assets componentes.

3.3.3 Descriptores Funcionales de los Mecanos

La búsqueda de elementos reutilizables concretos en una biblioteca de reutilización se realiza utilizando el potencial de clasificación que se incorpora. La clasificación es una parte crucial de la solución al problema de la localización de los elementos reutilizables deseados en el proceso de desarrollo con reutilización. Si éstos se almacenan en la biblioteca sin ninguna organización, no sólo será difícil encontrarlos, sino también comprenderlos.

El método de clasificación de los mecanos (*y de sus assets componentes*) debe ser ortogonal al modelo que se está construyendo y debe ser facilitado por los servicios de la biblioteca. Es deseable que pudieran existir varias clasificaciones que permitan potenciar los métodos de búsqueda utilizados por los desarrolladores con reutilización.

En este sentido, la información que se aporta a través de los atributos de las entidades que forman el Modelo de Mecano pueden derivar directamente en métodos de clasificación basados en:

- *Texto libre:* que utilizan un vocabulario no controlado[9].
- *Facetas:* introducidas por Prieto-Díaz [Prieto-Díaz and Freeman, 1987a], [Prieto-Díaz, 1989] y ampliamente adoptadas en muchos otros trabajos sobre reutilización [Sorumgard et al., 1993], [Karlsson, 1995]; éstas representan diferentes perspectivas, puntos de vista o dimensiones de un dominio particular, sintetizando la materia referente a los elementos reutilizables. Así, a los elementos reutilizables se le asignan categorías mediante la síntesis de tuplas de términos de facetas.

Una forma complementaria de clasificación de los mecanos es ofrecer información sobre la funcionalidad que recogen, esto es, sobre sus requisitos funcionales. Sin embargo, esta aproximación tiene la dificultad derivada de la enorme diversidad existente en los tipos de assets que, clasificados en el nivel de análisis, capturan los requisitos funcionales.

La importancia de la reutilización de los requisitos es algo de lo que se ha discutido en este mismo trabajo, pero cobra especial importancia cuando se trabaja con elementos reutilizables de grano grueso con representación de elementos de grano fino procedentes de todo el ciclo de vida del software, porque los requisitos suponen el punto de entrada ideal al resto de los componentes y, en concreto, los requisitos funcionales ofrecen una perspectiva de usuario final [Gomaa, 1995] muy útil para el desarrollador con reutilización.

El problema de representar la funcionalidad de los elementos reutilizables, especialmente requisitos funcionales, ha sido abordado en diferentes trabajos.

Una aproximación consiste en representar los elementos reutilizables de una biblioteca de reutilización mediante especificaciones formales que describan sus propiedades funcionales. Debido a que las especificaciones pueden ser arbitrariamente abstractas, se pueden centrar en aquellas propiedades del elemento reutilizable que sean más relevantes en una operación de recuperación. Esta aproximación la han llevado a cabo Mili et al. sobre la base de una biblioteca de especificaciones de funciones implementadas en Pascal [Mili et al., 1997].

Sin embargo, la forma más normal de representar los requisitos funcionales es en documentos de diferente naturaleza, utilizando el lenguaje natural como el principal, aunque no único, elemento de expresión. Por este motivo se utilizan representaciones alternativas para presentar y acceder a la información funcional recogida en los apartados y en los párrafos de dichos documentos. En la bibliografía especializada se pueden encontrar algunas aproximaciones al problema de la representación y tratamiento de los requisitos

[9] Los términos de indexación se derivan del texto informativo que acompaña a los elementos reutilizables, en lugar de elegirlos de un conjunto predeterminado (*vocabulario controlado).*

funcionales; entre otros cabe destacar: las redes de representación de conceptos [Clark and Porter, 1997], bases de datos que se actualizan automáticamente al estar enlazadas mediante *plug-ins* a un procesador de textos con el que se modifica el documento de requisitos del software [Silva, 1998] o las fachadas que recogen la información considerada relevante de un elemento software reutilizable [Jacobson et al., 1997].

En el Modelo de Mecano esta cuestión se aborda mediante **Descriptores Funcionales**. Un descriptor funcional es un contenedor de índices a representaciones textuales de aquellos requisitos funcionales que el desarrollador para reutilización ha querido destacar; de forma que cada una de estas representaciones de requisitos es un asset componente del mecano del tipo **DescripciónRequisitoFuncional**.

No es obligatoria la presencia de un descriptor funcional en todo mecano, aunque un mecano puede tener varios, a manera de múltiples vistas funcionales de un mismo mecano. A su vez, varios mecanos pueden compartir el mismo descriptor funcional (*ver figura 3.11*).

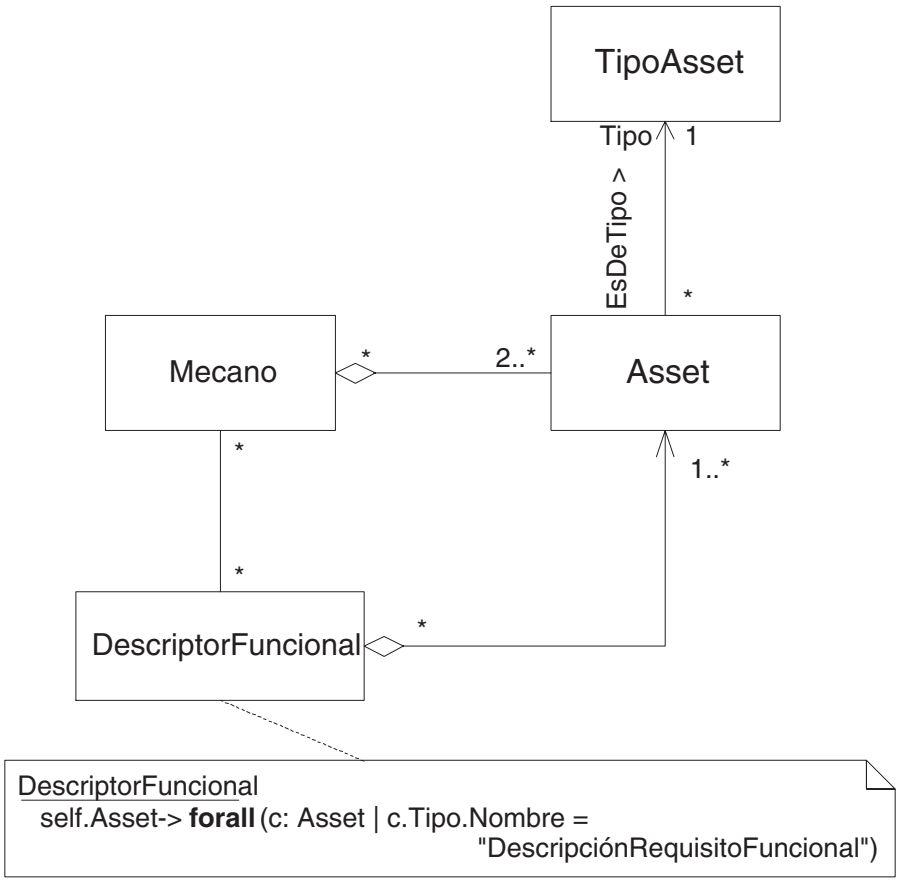

Figura 3.11: Descriptores funcionales en los mecanos.

La entidad DescriptorFuncional tiene los siguientes atributos:

- **Identificador:** Identificador único dentro del sistema.
- **FechaCreación:** Fecha de creación del descriptor funcional.
- **FechaModificación:** Fecha de última modificación del descriptor funcional.
- **Comentarios:** Texto con las notas descriptivas que el creador del descriptor quiera incluir.

Descripción de Componentes de un Descriptor Funcional

Se ha presentado un *descriptor funcional* como una colección de descripciones de requisitos funcionales que se encuentran especificados en el mecano asociado al descriptor. Sin embargo, un punto del cual va a depender directamente la utilidad de estos elementos es la representación que se utilice para describir dichos requisitos.

Dentro del modelo de reutilización que se está presentando, se ha optado por una solución que establece una plantilla o marco para la representación de los requisitos funcionales, utilizando el lenguaje natural como base para la expresión de éstos.

Esta plantilla para requisitos funcionales, que se ilustra en la tabla 3.3, es una variante de la presentada en los trabajos [Durán and Bernárdez, 1998] y [Durán and Bernárdez, 1999] sobre recolección de requisitos de un sistema software, donde se han eliminado algunos campos poco significativos para el proceso de reutilización.

La decisión para utilizar este formato de representación de los requisitos funcionales se justifica en los siguientes puntos:

- El lenguaje natural es la base de comunicación en la fase de elicitación de requisitos, lo cual lleva a que la forma más normal de documentar éstos sea utilizar algún mecanismo basado en lenguaje natural, que facilite el flujo de comunicación entre el desarrollador para reutilización y el desarrollador con reutilización.
- Estas plantillas se limitan a documentar requisitos, con independencia del paradigma de desarrollo que se haya seguido; sirven tanto para la documentación de casos de uso como de los requisitos que se reflejarán en los procesos de un DFD. Esto es debido a que no forman parte del modelo. Sin embargo, van a servir como puerta para llegar a los assets que representan los modelos de análisis en un mecano.

- Este formato de plantilla se adecua muy bien a su presentación en páginas **HTML**, lo cual facilita en gran medida la creación de sistemas de navegación de mecanos vía hipermedia.

- Su estructura tabular también se presta a buscar formatos de intercambio y almacenamiento en el repositorio, de forma que no sólo sea fácil su presentación sino también su tratamiento. En este sentido **XML** (*eXtensible Markup Language*) [Bray et al., 1998] parece el medio ideal para almacenar estas descripciones funcionales, siendo además extremadamente sencillo definir el **DTD** (*Documention Type Definition*) correspondiente a la gramática de las etiquetas para describir las entradas de la plantilla.

<identificador>	<nombre descriptivo>	
Descripción		
Secuencia normal	**Paso**	**Acción**
	1	
	2	
	3	
	4	
Excepciones	**Paso**	**Acción**
	1	
	2	
	3	
Comentarios		

Tabla 3.3. Plantilla para la descripción de los requisitos funcionales.

3.3.4 Modelo Semi-Formal de Mecano

Después del estudio de las diferentes partes que conforman la estructura de los elementos reutilizables de grano grueso denominados mecanos, sólo queda integrar todas las entidades en el Modelo de Mecano (*o modelo de repositorio para almacenar los mecanos*), expresado mediante un diagrama de clases de UML 1.x, tal y como aparece en la figura 3.12.

Los mecanos presentan 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 **Mecano**[10].

[10]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 3.12, ya que ésta 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

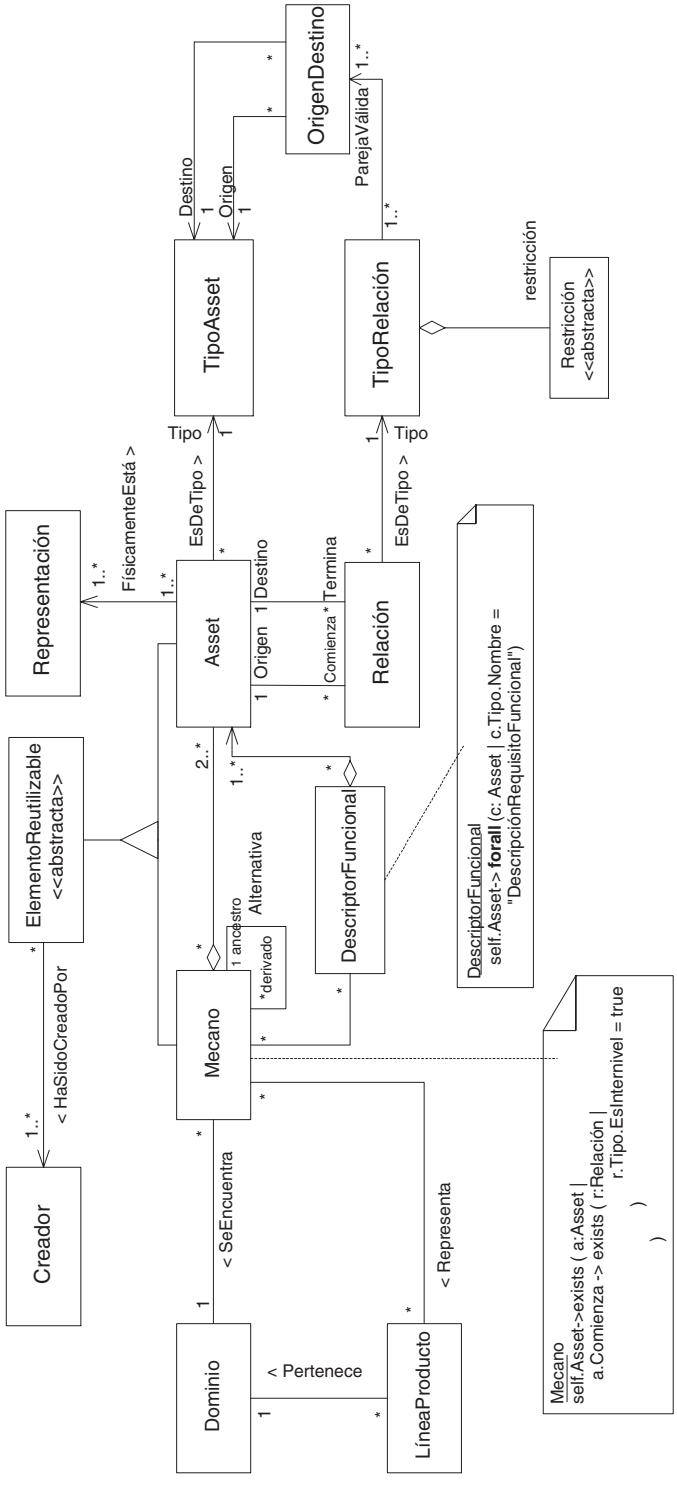

Figura 3.12: Modelo semi-formal de Mecano.

que haya relaciones internivel. Sin embargo, la sentencia OCL de la figura 3.12 se refiere a la necesidad de que exista al menos una relación internivel en un mecano, lo que ya implica la existencia de assets clasificados en al menos dos niveles de abstracción distintos.

El Modelo de Mecano, que se presenta en la figura 3.12, es el modelo de repositorio necesario para almacenar los mecanos, esto es, equivalente a un esquema de bases de datos, siendo el sistema gestor de bases de datos el repositorio o biblioteca de reutilización que los gestiona. El núcleo de este modelo es el componente reutilizable definido, que establece un mecano como un agregado de assets clasificados en diferentes niveles de abstracción relacionados entre sí por los enlaces definidos por las relaciones semánticas.

3.4 Modelo Formal de Mecano

Con el modelo semi-formal se ha obtenido una primera definición de los mecanos como estructuras de reutilización de grano grueso, así como el modelo de información necesario para su almacenamiento. Sin embargo, se requiere un modelo formal del modelo del componente reutilizable que permita obtener un proceso que pueda emplearse automáticamente para la verificación de los mecanos introducidos por un desarrollador para reutilización o para la generación por composición automática de nuevos mecanos en tiempo de reutilización.

Los mecanos podían ser considerados como un tipo especial de arquitectura, con la que se está estableciendo un conjunto de restricciones para definir elementos software reutilizables de grano grueso. En este sentido se puede afirmar que la teoría de grafos puede aportar la base necesaria para formalizar el Modelo de Mecano, donde los nodos representan los assets componentes del mecano y los arcos sus relaciones.

No obstante, la necesidad de distinguir los diferentes tipos de relaciones entre los assets componentes, derivados del conjunto de relaciones estructurales definido, lleva a la utilización de *grafos coloreados*, y más concretamente a un tipo especial de éstos denominados *grafos tubo*, para definir el modelo formal.

Tanto los grafos tubo como los grafos coloreados son entidades matemáticas que se emplean con asiduidad para modelar formalmente diseños arquitectónicos de sistemas software [Dean and Cordy, 1995], [Holt and Mancoridis, 1995], [Holt, 1996].

3.4.1 Introducción a los Grafos Coloreados

Los grafos coloreados, también conocidos como grafos tipados, son aquellos que sus vértices están interconectados por arcos coloreados dirigidos. El adjetivo de coloreado tiene el significado de que hay un conjunto finito de tipos de arcos (*cada uno de ellos denotado por un color*) de forma que cada arco del grafo tiene un color particular o tipo.

Formalmente un grafo coloreado es una tupla $(V, R_1, \ldots, R_n)$, donde V es el conjunto de vértices y $R_i \subseteq V \times V$ es el conjunto de aristas de color i.

3.4.2 Introducción a los Grafos Tubo

Un grafo tubo consiste en un árbol más un conjunto de aristas extra, llamadas tubos, entre los vértices del árbol. Estos grafos se utilizan en el modelado formal de diseños arquitectónicos de sistemas software cuando se imponen restricciones bien formadas a sus tubos para expresar las relaciones de conectividad entre los elementos software que forman el sistema.

El árbol representa el contenedor del subsistema software representado, mientras que los tubos representan las dependencias entre sus componentes.

La figura 3.13 muestra la estructura del sistema S que contiene dos subsistemas A y B. El subsistema A contiene un componente, C, mientras que el subsistema B contiene dos componentes, D y E. Además, se tienen sendas dependencias del componente C con respecto a los componentes de B. Esta situación se representa mediante una flecha de A a B, que aparece en la mitad de la figura 3.13 como un tubo hueco, a través del cual discurren los arcos de C a D y a E. En la parte de abajo del diagrama se muestra el grafo tubo que corresponde a la arquitectura del sistema S, donde el árbol de contención está representado por aristas sólidas, y las dependencias entre C y D y entre C y E, así como el tubo entre A y B, se representan por aristas punteadas. Todas las aristas punteadas son los tubos.

3.4.3 Mecanos Modelados como Grafos Tubo

Un mecano se formaliza mediante un grafo tubo que cumple una serie de restricciones. En primer lugar, la raíz del árbol de contención representa al elemento software reutilizable, esto es, al mecano en sí. Todo mecano está formado por dos o tres subsistemas, cada uno de los cuales representa un nivel de abstracción. A su vez, cada subsistema puede presentarse como un subárbol de contención de assets y un conjunto de tubos que representan las relaciones (*dependencias*) entre ellos. Además, existen tubos que representan las relaciones con otros assets clasificados en diferentes niveles de abstracción (*subsistemas*).

Los tubos equivalen a los enlaces semánticos entre los assets componentes del mecano, donde cada uno de ellos tiene un tipo (*un color o una etiqueta*) que expresa la relación estructural que define semánticamente al enlace.

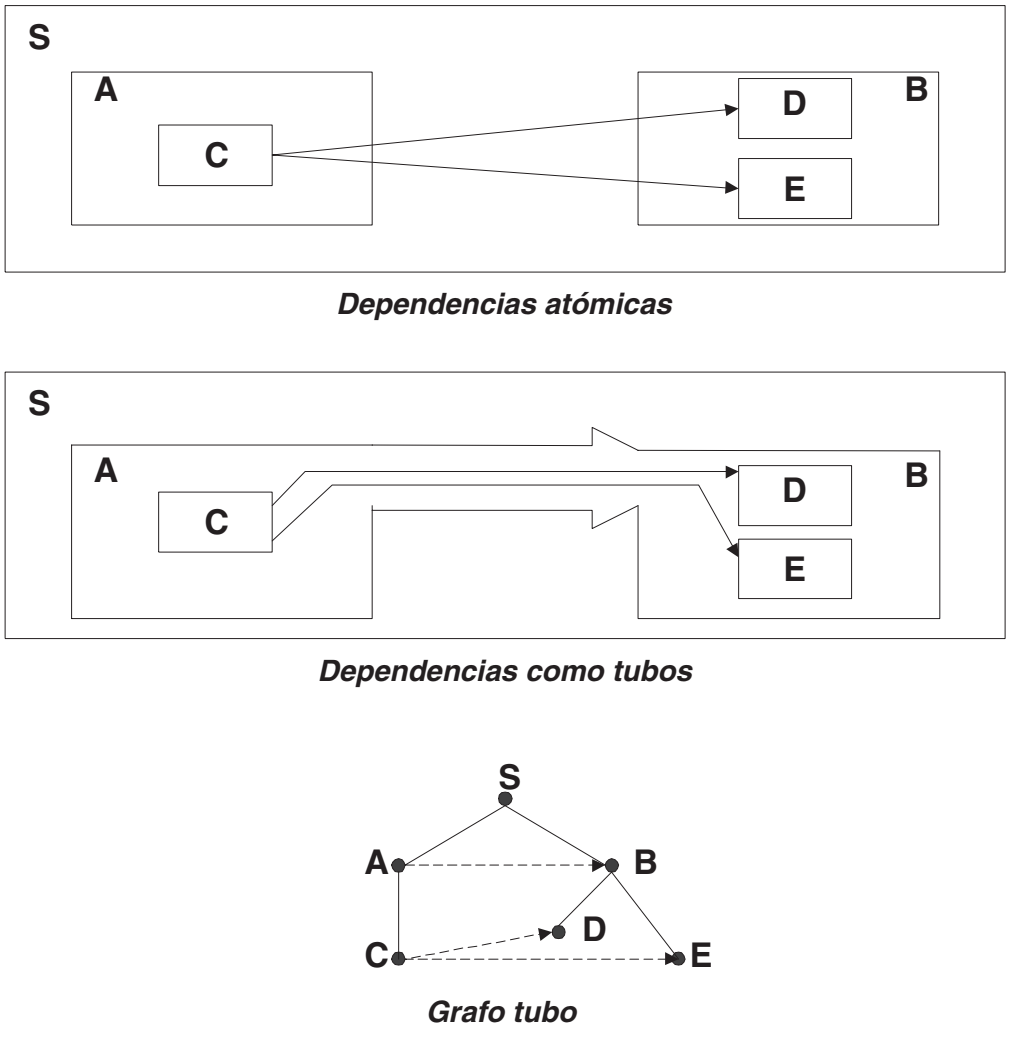

Figura 3.13: Uso de grafos tubo para modelar la estructura de un sistema software [Holt and Mancoridis, 1995].

Así pues, la estructura de grafo que da soporte a la definición estructural de mecano está formada por un conjunto de nodos, A, y dos conjuntos de aristas: el conjunto de las aristas de contención, C y el conjunto de los tubos, Tb.

El conjunto de nodos A está compuesto por:

- *La raíz del árbol de contención, que representa al propio mecano.*
- *Los niveles de abstracción presentes en el mecano; cada uno de los cuales aparece representado como un hijo de la raíz en el árbol de contención, denotándose por:*

n_1 = *Nivel de requisitos*
n_2 = *Nivel de diseño*
n_3 = *Nivel de implementación*

- *Los assets componentes del mecano, que aparecen representados como descendientes del correspondiente nivel de abstracción* n_i*, con* $i = 1..3$.

Esto es, el conjunto de los nodos del grafo tubo, A, se define como:

$$A = \{nodo\_raíz\} \cup N \cup \{Assets\} \tag{3.1}$$

donde:

$$N \subseteq \{n_1, n_2, n_3\} \tag{3.2}$$

$$|N| \geq 2 \tag{3.3}$$

Las aristas de contención, $C \subseteq A \times A$, representan por una parte *la relación existente entre la raíz del árbol de contención y los niveles de abstracción,* n_i *con* $i = 1..3$*, presentes en el mecano*, y por otra *la relación estructural de composición entre assets clasificados en el mismo nivel de abstracción.*

Los tubos pueden ser de seis tipos diferentes, expresan el resto de las relaciones estructurales: ***reificación, agregación, extensión, usa a, asociación fuerte*** y ***asociación débil***.

Se denota por T_R al conjunto de los distintos tipos de tubos que puede haber en un mecano:

$$T_R = \{\text{reificación, agregación, extensión, usa a, asociación fuerte, asociación débil}\} \tag{3.4}$$

Con el fin de simplificar las expresiones, a partir de ahora, se hará referencia a cada tipo de tubo mediante un índice, $i = 1..6$, en lugar de por su etiqueta, empezando por el índice $i = 1$, que hace referencia al tipo de tubo *reificación*, hasta el índice $i = 6$, que hace lo propio con el tipo de tubo *asociación débil.*

Se denota por T_i al conjunto de los tubos de tipo i de un mecano, de forma que:

$$T_i = \{(a_1, a_2, i) \mid \text{entre } a_1 \text{ y } a_2 \text{ existe un tubo de tipo } i, \text{ con } a_1, a_2 \in A \text{ e } i = \{1..6\}\} \tag{3.5}$$

$$T_i \subseteq A \times A \times \{i\} \tag{3.6}$$

Con la notación Tb, se hace referencia al conjunto de todos los tubos de un mecano, de forma que:

$$Tb = \bigcup_{i=1}^{6} \{(a_1,\, a_2,\, i) \mid \text{entre } a_1 \text{ y } a_2 \text{ existe un tubo de tipo } i, \text{ con } a_1,\, a_2 \in A \text{ e } i = \{1..6\}\} \tag{3.7}$$

$$Tb \subseteq A \times A \times T_R \tag{3.8}$$

Dado que existen diferentes tipos de relaciones (*tubos*) que pueden partir de cada uno de los assets nodos del grafo, se va a proceder a definir el grado de salida de un nodo del árbol de contención con respecto a una relación.

Definición 3.8 Grado de Salida de un Nodo del Árbol de Contención con Respecto a una Relación:

Se denomina grado de salida de un nodo a con respecto a una relación R a:

$$d(a,\, R) = |\, \{(a,\, y) \in A \times A \setminus a\, R\, y\}\, | \tag{3.9}$$

A continuación se van a enunciar las definiciones oportunas para la formalización del concepto de mecano mediante el uso de grafos tubo, para lo que previamente se van a establecer los siguientes predicados:

$$\mathbf{A}(p) \rightarrow \text{“El elemento } p \text{ pertenece al conjunto } A\text{”}$$

$$\mathbf{C}(x,\, y) \rightarrow \text{“Existe una arista de contención entre } x \text{ e } y;\, x \text{ es el padre de } y\text{”}$$

$$\mathbf{C_t}(x,\, y) \rightarrow \text{“Cierre transitivo de la relación C”}$$

$$\mathbf{T}(t,\, x,\, y) \rightarrow \text{“Existe una arista tubo } t \text{ entre } x \text{ e } y\text{”}$$

$$\mathbf{Raiz}(x) = \neg\, (\exists\, p)\, (\mathbf{A}(p) \wedge \mathbf{C}(p,\, x))$$

Definición 3.9 Árbol de Contención de un Mecano:

Dado un conjunto de nodos A y la relación de contención $C \subseteq A \times A$, se dice que $(A,\, C)$ es un árbol de contención de un mecano si y sólo si $(A,\, C)$ es un árbol, la raíz del árbol representa al mecano como elemento reutilizable, y el número de hijos de la raíz es dos o tres, cada uno de los cuales representa un nivel de abstracción presente en el mecano.

$$\begin{aligned} ArbolMecano\,(A,\, C) = {} & (\exists!\ r)\, (\mathbf{A}(r) \wedge \mathbf{Raiz}(r)) \wedge && (\textit{raíz única}) \\ & (d(r,\, C) \geq 2 \wedge d(r,\, C) \leq 3) \wedge && (\textit{hijos}) \\ & (\forall\, x)((\mathbf{A}(x) \wedge \neg\, \mathbf{Raiz}(x)) \Rightarrow (\exists!\, p)(\mathbf{A}(p) \wedge \mathbf{C}(p,\, x))) \wedge && (\textit{un padre}) \\ & (\forall\, p)\, (\forall x)((\mathbf{A}(p) \wedge \mathbf{A}(x) \wedge \mathbf{C_t}(p,\, x)) \Rightarrow (p \neq x)) && (\textit{no ciclos}) \end{aligned} \tag{3.10}$$

Además, las posibles configuraciones de niveles de abstracción presentes en el mecano (hijos de la raíz) son:

$$(\exists! r)((\mathbf{A}(r) \wedge \mathbf{Raiz}(r)) \wedge \begin{array}{l}((\mathbf{C}(r, n_1) \wedge \mathbf{C}(r, n_2) \wedge \mathbf{C}(r, n_3)) \vee \\ (\mathbf{C}(r, n_1) \wedge \mathbf{C}(r, n_2)) \vee \\ (\mathbf{C}(r, n_1) \wedge \mathbf{C}(r, n_3)) \vee \\ (\mathbf{C}(r, n_2) \wedge \mathbf{C}(r, n_3))))\end{array} \tag{3.11}$$

Donde:

n_1 = *Nivel de requisitos*
n_2 = *Nivel de diseño*
n_3 = *Nivel de implementación*

Definición 3.10 Arista Tubo Intranivel:

Una arista tubo intranivel es aquella que representa un tipo de relación estructural intranivel; estableciéndose entre dos assets descendientes del mismo nivel de abstracción n_i, con $i = 1..3$, dentro de un árbol de contención de un mecano $(A,\ C)$.

$$\begin{array}{rl}\mathbf{IntraNivel}(t) = & (\exists p)\,(\exists q)\,((\mathbf{A}(p) \wedge \mathbf{A}(q) \wedge \mathbf{T}(t, p, q)) \Rightarrow \\ & (\mathbf{Tb}(t) \wedge \mathbf{C_t}(n_i, p) \wedge \mathbf{C_t}(n_i, q) \wedge \\ & \quad (1 \leq i \leq 3)))\end{array} \tag{3.12}$$

Definición 3.11 Arista Tubo Internivel:

Una arista tubo internivel es aquella que representa un tipo de relación estructural internivel; estableciéndose entre dos assets descendientes de diferentes niveles de abstracción n_i, con $i = 1..3$, dentro de un árbol de contención de un mecano $(A,\ C)$.

$$\begin{array}{rl}\mathbf{InterNivel}(t) = & (\exists p)\,(\exists q)\,((\mathbf{A}(p) \wedge \mathbf{A}(q) \wedge \mathbf{T}(t, p, q)) \Rightarrow \\ & (\mathbf{T_1}(t) \wedge ((\mathbf{C_t}(n_i, p) \wedge \mathbf{C_t}(n_j, q)) \vee ((p = n_i) \wedge (q = n_j))) \wedge \\ & \quad (1 \leq i < j \leq 3)))\end{array} \tag{3.13}$$

Definición 3.12 Mecano:

Se dice que $M = (A,\ C,\ Tb)$ es un grafo con capacidad de representar mecanos si y sólo si $(A,\ C)$ es un árbol de contención y Tb contiene al menos un tubo de tipo reificación.

$$
\begin{array}{ll}
C \subseteq A \times A & (relacion\ de\ contencion) \\
Tb = \bigcup_{i=1}^{6} T_i \mid T_i \subseteq A \times A \times T_R & (aristas\ tubos) \\
T_1 \neq \emptyset & (existen\ reificaciones)
\end{array}
$$

De aquí en adelante se considerarán solamente los grafos tubo que cumplen las restricciones anteriores, y, salvo que se indique expresamente lo contrario, se van a denominar a estas estructuras mecanos.

Definición 3.13 PROFUNDIDAD DE UN ASSET EN UN ÁRBOL DE CONTENCIÓN DE UN MECANO:

Un asset $a \in A$, componente de un mecano $M = (A, C, Tb)$, tiene una profundidad n si tiene n antecesores propios en el árbol de contención.

$$Profundidad(a) = \mid \{p \in A \setminus p\, C^{+}\, a\} \mid \tag{3.14}$$

Donde C^{+} es el cierre transitivo de la relación C.

3.4.4 Restricciones en las Aristas Tubo

Se ha definido un mecano como un grafo tubo formado por un árbol de contención y un conjunto de aristas tubo entre sus assets componentes, con la restricción de que debe existir al menos una relación de reificación. Si además se imponen unas restricciones adecuadas sobre las aristas tubo se obtendrán mecanos bien formados.

Se considera que una arista tubo intranivel $(x, y, i) \in T_i$, con $i = 2..6$, está bien formada si satisface una de las siguientes reglas:

- **(R1)** *Regla de hermandad entre componentes:* x e y son componentes dentro del mismo compuesto (*hermanos propios* $x \neq y$); con x e y clasificados en el mismo nivel de abstracción.
- **(R2)** *Regla de herencia de padres a hijos:* El padre de x tiene una arista tubo intranivel al padre de y.
- **(R3)** *Regla de dependencias transitivas:* El padre de x tiene una arista tubo intranivel a y.
- **(R4)** *Regla de dependencia de un componente:* x tiene una arista tubo intranivel al padre de y.

Se considera que una arista tubo internivel $(x,\ y,\ 1)\ \in\ T_1$, está bien formada si se cumple una de las siguientes reglas:

- **(R5)** *Regla de reificación de los niveles de abstracción:* Los assets que representan los niveles de abstracción de un mecano se relacionan mediante tubos internivel.

- **(R6)** *Regla de reificación por pares:* El padre de x tiene un tubo del tipo *reificación* al padre de y, con x e y clasificados en diferentes niveles de abstracción.

- **(R7)** *Regla de reificación transitiva:* El padre de x tiene un tubo internivel a y; con x e y clasificados en diferentes niveles de abstracción.

- **(R8)** *Regla de reificación a un componente:* x tiene un tubo internivel al padre de y; con x e y clasificados en diferentes niveles de abstracción.

En la figura 3.14 se ofrece una representación visual del conjunto de tubos permitidos. Los arcos etiquetados representan las relaciones en un mecano. Los arcos punteados sin etiquetar representan los tubos permitidos. Es importante recalcar, que mientras que en las cuatro primeras reglas, $R1-R4$, se está haciendo referencia a los tubos intranivel, en las cuatro siguientes, $R5-R8$, se hace lo propio con los tubos internivel, los cuales representan a la relación de reificación.

Se denota por $T_P(M)$ al conjunto de tubos permitidos de un mecano $M\ =\ (A,\ C,\ Tb)$, aunque cuando el contexto sea obvio se utilizará T_P en lugar de $T_P(M)$.

Definición 3.14 Tubos Permitidos:

Dado un mecano $M\ =\ (A,\ C,\ Tb)$*, su conjunto de tubos permitidos* (T_P) *es:*

$$T_P\ =\ \{(x,\ y,\ i)\ \in\ A\ \times\ A\ \times\ T_R\ |\ \ \mathbf{TuboBF}(x,\ y)\} \qquad (3.15)$$

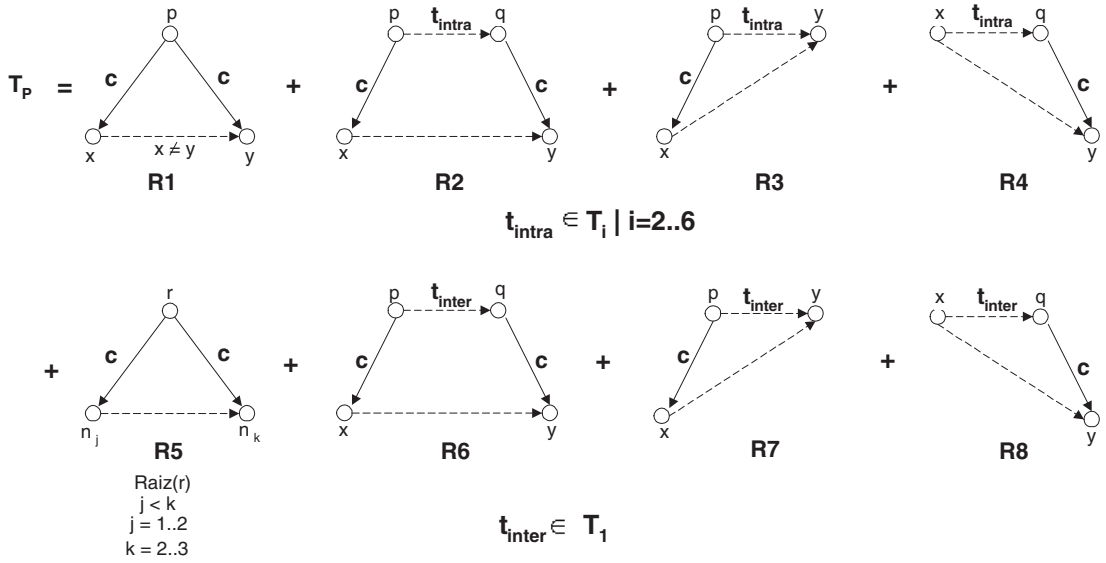

Figura 3.14: Representación visual de los tubos permitidos.

Donde:

$$
\begin{aligned}
\mathbf{TuboBF}(x,\,y) = (\exists\, p)\,(\exists\, q)\;\; & (\mathbf{A}(p) \wedge \mathbf{A}(q) \wedge \\
& ((\neg\, \mathbf{Raiz}(p) \wedge \mathbf{C}(p,\,x) \wedge \mathbf{C}(p,\,y) \wedge x \neq y) \vee && \textit{(R1)} \\
& (\mathbf{C_t}(n_j,\,p) \wedge \mathbf{C_t}(n_j,\,q) \wedge \mathbf{C}(p,\,x) \wedge \mathbf{C}(q,\,y) \wedge \\
& \quad \mathbf{T}(t,\,p,\,q) \wedge \mathbf{T_i}(t)) \vee && \textit{(R2)} \\
& (\mathbf{C_t}(n_j,\,p) \wedge \mathbf{C_t}(n_j,\,y) \wedge \mathbf{C}(p,\,x) \wedge \mathbf{T}(t,\,p,\,y) \wedge \mathbf{T_i}(t)) \vee && \textit{(R3)} \\
& (\mathbf{C_t}(n_j,\,x) \wedge \mathbf{C_t}(n_j,\,q) \wedge \mathbf{C}(q,\,y) \wedge \mathbf{T}(t,\,x,\,q) \wedge \mathbf{T_i}(t)) \vee && \textit{(R4)} \\
& (\mathbf{Raiz}(r) \wedge \mathbf{C}(r,\,p) \wedge \mathbf{C}(r,\,q) \wedge (p = n_k) \wedge (q = n_l)) \vee && \textit{(R5)} \\
& (((\mathbf{C_t}(n_k,\,p) \wedge \mathbf{C_t}(n_l,\,q)) \vee ((n_k = p) \wedge (n_l = q))) \wedge \\
& \quad \wedge \mathbf{C}(p,\,x) \wedge \mathbf{C}(q,\,y) \wedge \mathbf{T}(t,\,p,\,q) \wedge \mathbf{T_1}(t)) \vee && \textit{(R6)} \\
& (\mathbf{C_t}(n_k,\,p) \wedge \mathbf{C_t}(n_l,\,y) \wedge \mathbf{C}(p,\,x) \wedge \mathbf{T}(t,\,p,\,y) \wedge \mathbf{T_1}(t)) \vee && \textit{(R7)} \\
& (\mathbf{C_t}(n_k,\,x) \wedge \mathbf{C_t}(n_l,\,q) \wedge \mathbf{C}(q,\,y) \wedge \mathbf{T}(t,\,x,\,q) \wedge \mathbf{T_1}(t)))) && \textit{(R8)}
\end{aligned}
\tag{3.16}
$$

Con $1 \leq j \leq 3$, $1 \leq k < l \leq 3$ *e* $i = 2..6$.

Definición 3.15 Mecano Bien Formado:

Un mecano $M = (A,\, C,\, Tb)$ *es un mecano bien formado si y sólo si todas sus aristas tubo* Tb *forman un subconjunto de su conjunto de aristas tubo permitidas* T_P*:*

$$Tb \subseteq T_P \tag{3.17}$$

Lo cual puede expresarse de la siguiente forma:

$$\mathbf{BF}(M) = (\forall\, t)\,((\exists\, x)(\exists\, y)(\mathbf{A}(x) \wedge \mathbf{A}(y) \wedge \mathbf{T}(t,\, x,\, y) \wedge \mathbf{Tb}(t) \wedge \mathbf{TuboBF}(x,\, y))) \tag{3.18}$$

Definición 3.16 Todos los Mecanos Bien Formados:

Se define ν como el conjunto de todos los mecanos bien formados.

$$\nu = \{M \mid \mathbf{BF}(M)\} \tag{3.19}$$

Capítulo 4

Construcción de Mecanos

Según se ha dejado intuir en el *Capítulo 3*, la construcción de mecanos, dentro del modelo de reutilización que se está definiendo, no se limita a un enfoque compositivo único propio del desarrollo para reutilización, sino que se aporta un nuevo enfoque en el que los mecanos pueden construirse de forma automática en tiempo de reutilización sobre la base de los elementos reutilizables ya existentes en el repositorio.

La introducción de esta perspectiva de creación de mecanos mediante la composición automática de elementos reutilizables, se acerca a un enfoque de generación en el proceso de reutilización, cayendo dentro de los denominados *generadores basados en composición* [Biggerstaff, 1999].

Así pues, dentro del modelo de reutilización se habría conseguido reunir las dos tendencias principales en la reutilización del software, la composición y la generación, ofreciendo una aproximación mixta o híbrida que va a aportar una mayor flexibilidad al proceso global de reutilización, donde no sólo se crean los elementos reutilizables en la parte de la ingeniería del dominio o desarrollo para reutilización, sino que también se van a crear automáticamente en tiempo de reutilización.

La creación de mecanos mediante una composición o extracción automática de assets existentes en el repositorio, es un camino que aporta una aproximación a la solución buscada por el desarrollador con reutilización, que le servirá como punto de partida cuando no existan mecanos en el repositorio que satisfagan sus necesidades. Como se discutirá más adelante, para obtener las mejores aproximaciones en la construcción automática, se debe contar, además de con el mecanismo que realice dicha operación, con una organización adecuada dentro del proceso de reutilización, que haga posible sacar el máximo partido de la flexibilidad que se está aportando.

El resto del capítulo se organiza en dos apartados principales. El primero de ellos aborda la creación de mecanos como una actividad del desarrollo para reutilización, es decir, como una composición manual de elementos reutilizables de grano fino. En el apartado dos, se presenta la creación de mecanos en tiempo de reutilización gracias a un proceso de composición automática guiado por el desarrollador con reutilización, en el que se intenta sacar el mayor provecho de la información que aporta el propio Modelo de Mecano y de la política de organización de la reutilización puesta en marcha en el organismo.

4.1 Creación de Mecanos en el Desarrollo Para Reutilización

Como ya se había argumentado los mecanos deben integrarse en el proceso general de reutilización. En consecuencia, como cualquier elemento reutilizable, los mecanos deben ser diseñados y construidos por los desarrolladores para reutilización en las actividades propias de la ingeniería de dominio, para su futura reutilización en la ingeniería de aplicación.

El diseño de los mecanos viene condicionado por el uso que el organismo que gestiona la biblioteca de reutilización quiera hacer de éstos elementos reutilizables, apartado que se detalla en la parte organizativa del modelo de reutilización, el cual se discute en el *Capítulo 5*. La única restricción que se impone por definición es que los mecanos deben reflejar un refinamiento en cuanto al nivel de abstracción de sus componentes.

El desarrollo de los assets componentes de un mecano es una actividad que esta propuesta de tesis considera perfectamente tratada en las diferentes guías ya existentes sobre el tema, véase [NATO, 1992b], [Karlsson, 1995], [García et al., 1997a].

El proceso de construcción *manual* de los mecanos debe permitir construir la estructura de grafo, especificada en la *definición 3.12*, a partir de la información almacenada en el repositorio. Dicho proceso de construcción debe ser tal que permita, además, verificar que el mecano que se construye es un mecano bien formado según la *definición 3.15*.

Las directrices de este proceso de construcción/verificación se resumen en los siguientes pasos:

1. Construir el árbol de contención del mecano.
   (a) Especificar la raíz, constituida por un nodo que represente al mecano (*equivale a introducir los datos requeridos por la entidad* ***Mecano*** *del Modelo de Mecano*).
   (b) Introducir un hijo de la raíz por cada uno de los niveles de abstracción presentes en el mecano.

(c) Para cada nivel de abstracción ir añadiendo los assets componentes.

(d) Para cada asset compuesto, ir construyendo los niveles inferiores del árbol.

2. Establecer los tubos intranivel.

3. Establecer los tubos internivel.

Para cuidar que el proceso de construcción de mecanos produzca como resultado mecanos bien formados se ha creado un gramática de grafos que dicte el buen discurrir de la construcción. Las gramáticas de grafos son herramientas formales que se utilizan con asiduidad para la descripción de arquitecturas software [Degano and Montanari, 1987], [Le Métayer, 1996], [Le Métayer, 1998], [Hirsch et al., 1998].

4.1.1 Gramática de Grafos para Mecanos Bien Formados

La construcción de un mecano mediante la composición de assets puede verse como la construcción de un grafo a partir de derivaciones dictadas por un conjunto de reglas establecidas. En este sentido se puede establecer una gramática de grafos para mecanos bien formados que de forma declarativa especifique las condiciones necesarias para considerar que un mecano está bien formado.

Una *gramática de grafos dependiente del contexto* $GDC = (NF, F, P, S)$ [Fenton and Hill, 1993] consiste en un conjunto NF aristas no terminales, un conjunto F de aristas terminales, un conjunto P de reglas dependientes del contexto y de la arista no terminal inicial $S \in NF$. La parte izquierda de una regla dependiente del contexto es cualquier grafo que contenga al menos una arista no terminal de NF, y posiblemente alguna arista terminal de F. La parte derecha de una regla dependiente del contexto puede ser cualquier grafo con arista no terminales de NF y terminales de F, cumpliéndose que el número de aristas (*terminales y no terminales*) del grafo del lado derecho de la regla de producción no es menor que las aristas del grafo del lado izquierdo. Un grafo es terminal si todas sus aristas son terminales.

Dada una gramática de grafos dependiente del contexto GDC, todos los grafos terminales, Γ, que se derivan de la arista inicial S, forman el lenguaje $L(GDC)$ generado por la gramática GDC.

$$L(GDC) = \{\Gamma \mid S \stackrel{*}{\Rightarrow}, \ y \ \Gamma \ es \ terminal\} \tag{4.1}$$

Donde $\stackrel{*}{\Rightarrow}$ denota el cierre transitivo de la derivación directa.

En las figuras 4.1, 4.2 y 4.3 se recoge la gramática G_M dependiente de contexto completa para los mecanos bien formados.

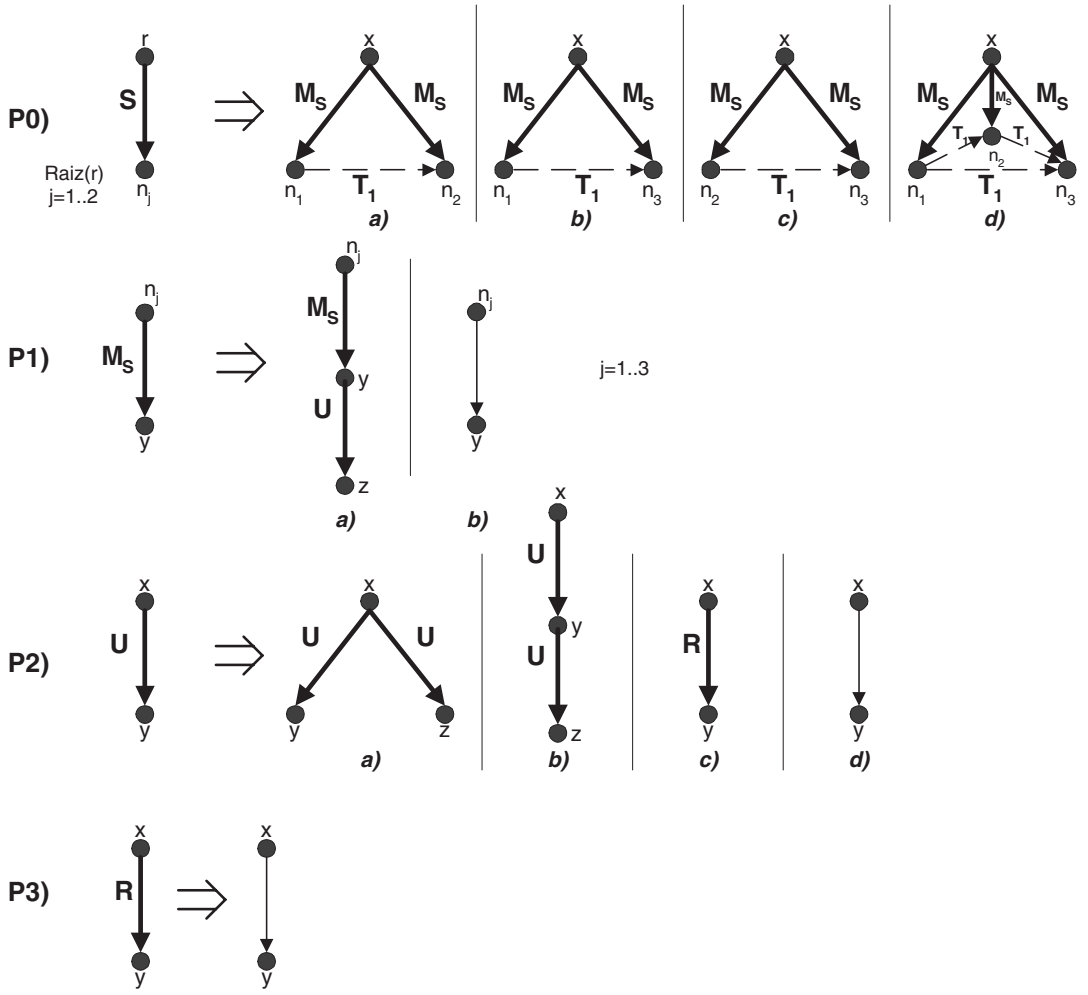

Figura 4.1: Reglas para la construcción del árbol de contención del mecano.

La gramática G_M tiene once reglas de producción. Las reglas $P0a - P0d$ se ocupan de crear la raíz del árbol de contención del mecano y sus hijos, es decir el mecano como elemento reutilizable y los niveles de abstracción que están presentes en él. Las reglas de producción $P1 - P2$ terminan de crear el árbol de contención del mecano, introduciendo los assets en cada uno de los niveles de abstracción y creando los niveles del árbol necesarios para la representación de assets agregados y compuestos. Las reglas $P4 - P7$ se corresponden con las cuatro reglas para la correcta formación de los tubos intranivel, $R1 - R4$, mientras que las reglas $P8 - P10$ se corresponden con las reglas $R6 - R8$ para la creación

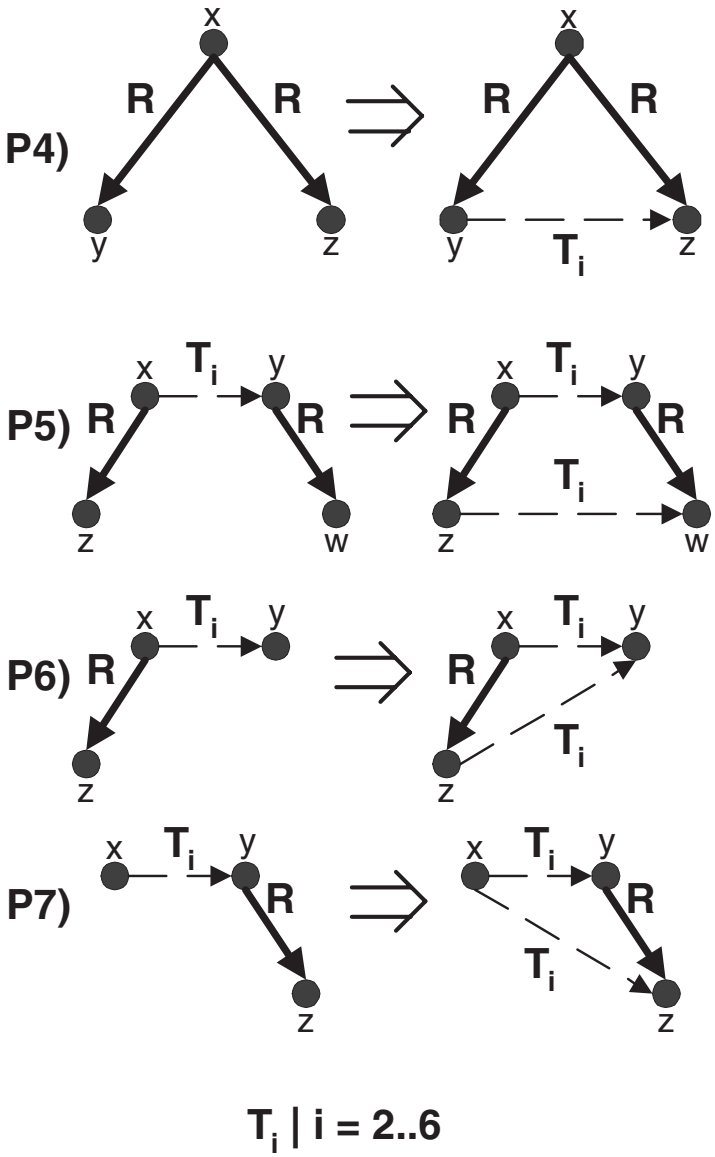

Figura 4.2: Reglas para la construcción de los tubos intranivel.

de los tubos internivel[1]. La regla $P3$ simplemente transforma aristas de contención no terminales en terminales.

La arista no terminal S se encuentra en el lado izquierdo de la regla $P0$, haciendo las veces de axioma. Las etiquetas S, M_S, U y R representan aristas de contención no terminales, mientras que las etiquetas T_i, con $i = 1..6$, representan tubos de tipo i, donde

[1] La regla $R5$ se encuentra embebida en las producciones $P0a - P0d$.

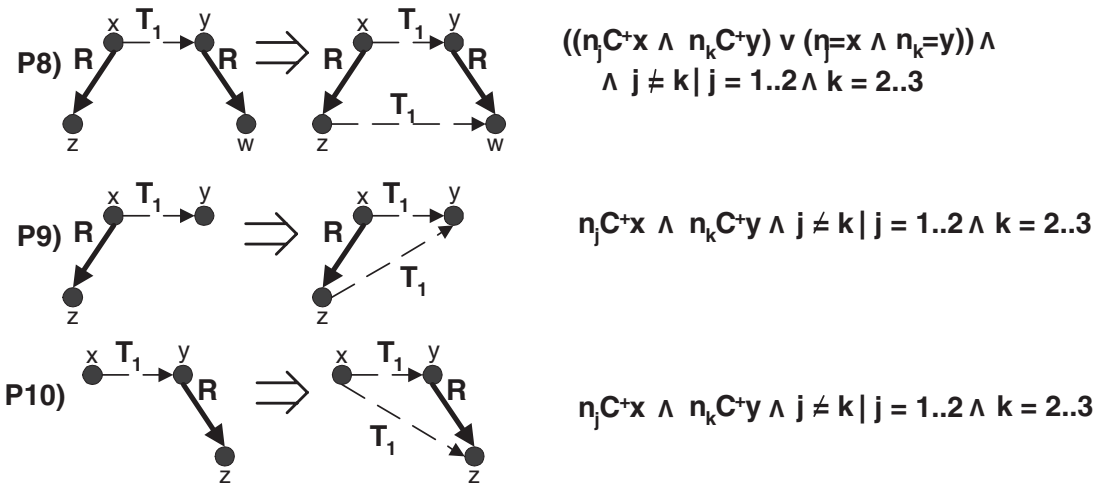

Figura 4.3: Reglas para la construcción de los tubos internivel.

$i = 1$ hace referencia a los *tubos internivel* e $i = 2..6$ a los *tubos intranivel*.

Existen tres tipos de nodos (*todos pertenecientes al conjunto* A), la raíz, los niveles de abstracción n_i, con $i = 1..3$, y los assets componentes del mecano. Estas distinciones posibilitan que todo mecano que se construya a partir de la gramática G_M esté bien formado con respecto a las restricciones de los tubos intranivel e internivel, además de cumplir las restricciones impuestas por definición, de modo que todo mecano tenga más de un nivel de abstracción y alguna relación intranivel.

Del estudio de la gramática G_M se entiende que hay una clara separación entre la raíz y el nivel uno del árbol de contención, y el resto de niveles donde están los assets componentes (*aristas* M_S *y* U). Las aristas de contención aparecen representadas en las figuras 4.1, 4.2 y 4.3 con trazo continuo y grueso.

Para que un tubo, independientemente de su tipo, pueda establecerse, sus aristas de contención deben haber sido previamente transformadas a aristas R utilizando la regla de producción *P2c*; esto es así porque gramáticas que permitiesen expandir aristas después de estar conectadas por tubos darían lugar a mecanos mal formados.

Todas las aristas terminales están sin etiquetar, las aristas terminales de contención se han representado con trazo continuo y fino, mientras que para las aristas tubo se ha recurrido a trazo discontinuo y fino.

A continuación, se va probar que la gramática G_M permite construir mecanos bien formados, dictando esta gramática el orden canónico para la construcción de mecanos.

Proposición 4.1

Cualquier mecano bien formado está en $L(G_M)$.

Se va a probar que cualquier mecano bien formado $M = (A,\ C,\ Tb)$ puede construirse utilizando la gramática dependiente del contexto G_M:

$$\nu \subseteq L(G_M) \tag{4.2}$$

Sea M un mecano bien formado. Por la *definición 3.15* se tiene que todo tubo $t \in Tb$ verifica que $t \in T_P$. Esto significa que el tubo t debe cumplir una de las ocho reglas $R1 - R8$.

Cada una de estas reglas tiene una regla de producción equivalente en la gramática G_M:

- $R1 - R4 \rightarrow P4 - P7$
- $R5 \rightarrow P0$
- $R6 - R8 \rightarrow P8 - P10$

Además, como la gramática G_M ha sido construida de forma que cualquier tubo sólo se puede introducir mediante las reglas $P0$, $P4 - P7$, $P8 - P10$, cualquier mecano, M, construido utilizando G_M sólo pueden tener tubos bien formados, y por lo tanto ser un mecano bien formado, $\mathbf{BF}(M)$.

Si existiese un mecano bien formado $M = (A,\ C,\ Tb)$, tal que $M \notin L(G_M)$ implicaría que es un mecano que no se podría construir utilizando la gramática G_M, por lo que existiría al menos un tubo $t \in Tb$ que no podría ser construido con una de las reglas de producción de la gramática $P0$, $P4 - P7$, $P8 - P10$. Esta situación conduce a que t no satisface una de las reglas $R1 - R8$, de lo que se deduce que $t \notin T_P$ y por tanto M no puede ser un mecano bien formado.

Lo cual prueba que $\nu \subseteq L(G_M)$.

□

Para corroborar la proposición 4.1 se va a proceder a explicar como se construye un mecano utilizando las reglas de producción de la gramática G_M.

En primer lugar se crea el árbol de contención del mecano. Se comienza con la identificación del nodo raíz, r, que representa al mecano como elemento reutilizable. A continuación

se añaden sus hijos, cada uno de los cuales representa un nivel de abstracción, para lo cual se aplica la regla de producción $P0$ que mejor se ajuste al mecano que se quiera construir, de forma que después de su aplicación el grado de salida de r con respecto a la relación C, $d(r,\, C)$, sea igual al número de niveles de abstracción contemplados en el mecano.

Una vez que se han creado los niveles de abstracción se aplica la regla *P1a* para extender cada uno de los niveles de abstracción presentes.

Cada rama se va a completar aplicando la regla de producción *P2a* tantas veces como indique el grado de salida de cada nivel n_i menos uno, $d(n_i,\, C) - 1$, con $i = 1..3$, dando por supuesto que $d(n_i,\, C) > 0$.

Para completar el árbol de contención se aplican recursivamente las reglas *P2b* para crear un nuevo nivel de profundidad y *P2a* para llenarlo.

En la figura 4.4 se muestran los pasos para la construcción del árbol de contención del mecano.

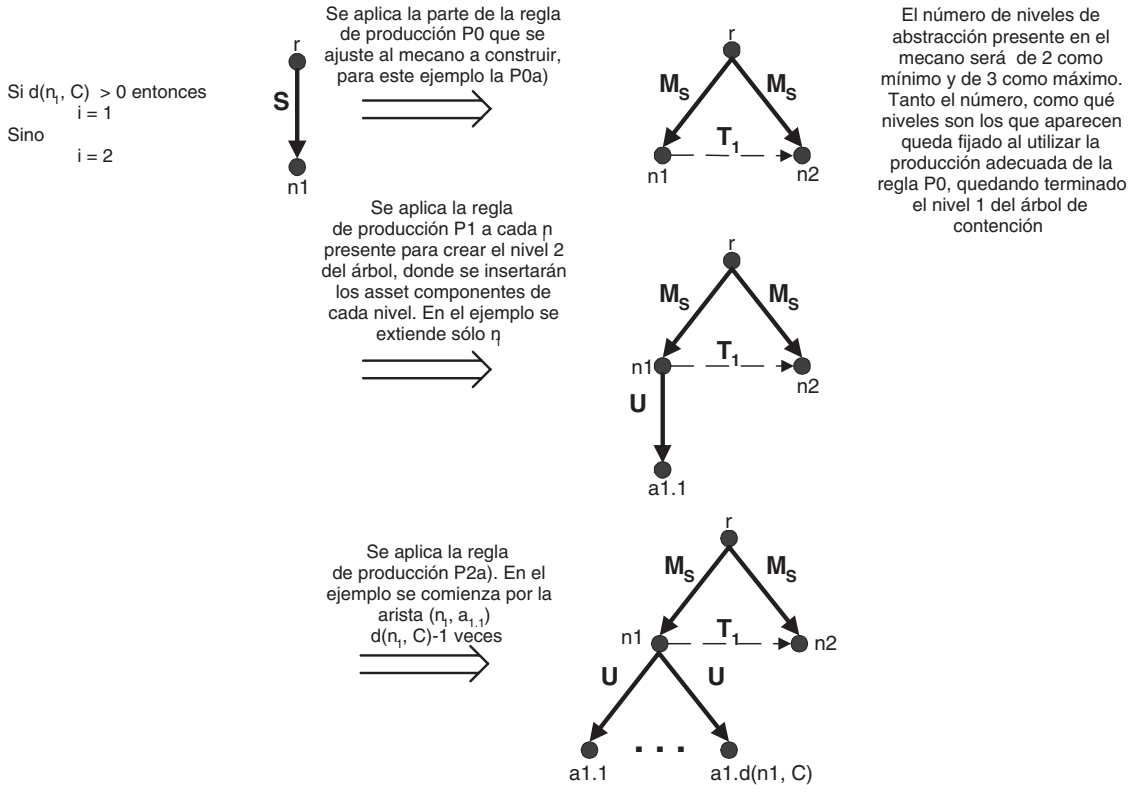

Figura 4.4: Derivación del árbol de contención de un mecano.

Como paso previo a la introducción de las aristas tubo, y una vez que el árbol de

contención ha sido construido, se debe aplicar la regla de producción *P2c* para convertir las aristas U en aristas R. Dado que la raíz del árbol de contención, r, no puede tener tubos, y los niveles de abstracción, n_i de un mecano bien formado ya los ha recibido en la construcción inicial, se comienza con los $d(n_i, C)$ hijos de cada nivel de abstracción n_i presente en el mecano, con $i = 1..3$, que son assets de segundo nivel dentro del mecano. Para cada nivel de profundidad np del árbol, con $np \geq 2$, se hace lo siguiente:

1. Para cada asset a_i del nivel np, se introducen todos sus tubos intranivel. Se comienza por seleccionar el asset a_j, tal que el tubo $(a_i, a_j, k) \in T_k$, con $k = 2..6$, no ha sido creado todavía y $profundidad(a_j) - profundidad(a_i)$ es mínima. Se continúa hasta que todas las aristas tubo de cada tipo k hayan sido creadas para el asset a_i. Para que el mecano esté bien formado, cada tubo $(a_i, a_j, k) \in T_k$, con $k = 2..6$, debe satisfacer una de las siguientes reglas:

   - **(R1):** a_i y a_j tienen el mismo padre, utilizándose la regla de producción $P4$, o
   - **(R2):** Existe un tubo intranivel $t \in Tb$ entre los padres de a_i y a_j, empleándose la regla de producción $P5$, o
   - **(R3):** Existe un tubo intranivel $t \in Tb$ entre el padre de a_i y a_j, haciendo uso de la regla de producción $P6$, o
   - **(R4):** Existe un tubo intranivel $t \in Tb$ entre a_i y el padre de a_j, recurriéndose entonces a la regla de producción $P7$.

2. Para cada asset a_i del nivel de profundidad np, que sea descendiente del nivel de abstracción de requisitos, n_1, o del nivel de abstracción de diseño, n_2, se introducen todos sus tubos internivel; de forma que cada $(a_i, a_j, 1) \in T_1$, debe haber sido creado utilizando alguna de las siguientes reglas, habiéndose comenzado por seleccionar a aquellos a_j tales que $profundidad(a_j) - profundidad(a_i)$ es mínima:

   - **(R6):** Existe un tubo internivel $t \in T_1$ entre los padres de a_i y a_j, pudiéndose dar el caso que el padre de a_i fuera n_k, $n_k \, C \, a_i$, y el padre de a_j fuera n_l, $n_l \, C \, a_j$, con $1 \leq k < l \leq 3$; empleándose la regla de producción $P8$, o
   - **(R7):** Existe un tubo internivel $t \in T_1$ entre el padre de a_i y a_j, de forma que se cumpla que $n_k \, C^+ a_i \wedge n_l \, C^+ a_j$, con $1 \leq k < l \leq 3$; empleándose la regla de producción $P9$, o
   - **(R8):** Existe un tubo internivel $t \in T_1$ entre a_i y el padre de a_j, cumpliéndose que $n_k \, C^+ a_i \wedge n_l \, C^+ p \wedge p \, C \, a_j$ con $1 \leq k < l \leq 3$; empleándose la regla de producción $P10$.

Aplicando las reglas de producción oportunas $P4 - P10$, se puede crear cualquier tubo bien formado $t = (a_i, a_j, k)$, tal que $t \in T_k$, con $k = 1..6$.

Después de haber creado todos las aristas tubo del mecano, se aplican las reglas de producción $P1b$ y $P3$, según se preste, para convertir todas las aristas en terminales.

Proposición 4.2

Cualquier grafo perteneciente a $L(G_M)$ es un mecano bien formado.

Se va a probar que cualquier grafo $\Gamma = (A, C, Tb)$ construido utilizando la gramática G_M es un mecano bien formado:

$$L(G_M) \subseteq \nu \tag{4.3}$$

Si un grafo Γ ha sido construido utilizando la gramática G_M, $\Gamma \in L(G_M)$, sólo pueden haberse utilizado las reglas de producción de la gramática G_M.

Esto conduce a que el árbol de contención (A, C) del grafo Γ es un árbol de contención de un mecano, $ArbolMecano(A, C)$. Si esto es cierto, cualquier árbol de contención que pueda crearse aplicando la gramática G_M debe ser un árbol, y el número de hijos de la raíz debe ser 2 ó 3.

Para demostrar que (A, C) es un árbol se puede recurrir a la propiedad de los árboles por la que éstos se pueden considerar como grafos conexos y el número de sus aristas es una unidad menor que el número de sus nodos.

Como las únicas producciones de G_M que se ocupan de la construcción del árbol de contención son $P0 - P2$, y ninguna de ellas da lugar a vértices no conectados, el resultado es un grafo conexo. Además, se comienza con una única arista conectada por dos vértices, al construir el nivel uno del árbol de contención se aplica la regla de producción $P0$, de forma que cualquiera que sea la parte utilizada se introducen tantas aristas de contención nuevas como vértices nuevos, por lo que la invariante se sigue cumpliendo. La extensión y llenado de niveles involucra a las reglas $P1a$, $P2a$ y $P2b$ que introducen una arista y un vértice. Tanto $P1b$ como $P2c$ y $P2d$ dejan invariante el número de aristas y vértices. De aquí se puede deducir que el número de aristas en el grafo Γ es siempre una unidad menor que el número de vértices.

Para demostrar que el número de hijos de la raíz de cualquier árbol de contención construido es 2 ó 3, basta con comprobar que el primer nivel del árbol sólo se puede crear con la regla de producción $P0$, operación que se realiza partiendo del axioma inicial.

Cualquiera de las partes de la regla $P0$ conduce a que el número de hijos de la raíz es 2 ó 3, donde la raíz representa al mecano como elemento reutilizable y cada uno de sus hijos un nivel de abstracción presente en el mecano.

Por tanto:

$$\text{Si } \Gamma \in L(G_M) \text{ entonces } ArbolMecano(A, C)$$

Por otra parte, y una vez que se tiene construido el árbol de contención del grafo Γ, para poder considerar que $\mathbf{BF}(\Gamma)$, cualquier arista tubo establecida entre dos elementos cualquiera x y y, con $x, y \in A$, tiene que cumplir que es una arista tubo bien formada, $\mathbf{TuboBF}(x, y)$.

Como las reglas de producción $P0$ y $P4 - P10$ son las únicas que introducen tubos, todos los tubos $(x, y, k) \in Tb$, con $k = 1..6$, creados con la gramática G_M se derivan de ellas.

Al crearse el nivel 1 del árbol de contención con la regla $P0$ se crean los tubos internivel entre estos niveles, que son los únicos permitidos, pues ninguna otra regla puede introducir tubos intra o internivel en este nivel (*Regla de reificación de los niveles de abstracción*). Si se utiliza la regla $P4$, x e y tienen el mismo padre (*Regla de hermandad entre componentes*). Si se emplea la regla $P5$, los padres de x e y tienen un tubo entre ellos (*Regla de herencia de padres a hijos*). Si se hace uso de la regla $P6$, hay una arista tubo entre el padre de x e y (*Regla de dependencias transitivas*). Si $P7$ es seleccionada, existe un tubo entre x y el padre de y (*Regla de dependencia de un componente*). Las reglas $P8 - P10$ son las equivalentes a las reglas $P5 - P7$ pero para tubos internivel. En cualquier caso, los tubos derivados satisfacen las reglas de formación.

Por tanto:

$$\text{Si } \Gamma \in L(G_M) \text{ entonces } \mathbf{BF}(\Gamma)$$

Con lo que se prueba que $L(G_M) \subseteq \nu$.

□

Teorema 4.1

El conjunto de todos los mecanos bien formados, ν, es el lenguaje generado por la gramática dependiente del contexto G_M.

$$\nu \equiv L(G_M) \tag{4.4}$$

La prueba a este teorema se tiene en las proposiciones 4.1 y 4.2.

Una regla obvia en la construcción de mecanos es que dos assets que están relacionados por una relación de contención (*composición*) no pueden estar relacionados por otra relación estructural. Esto se puede extender a los assets lógicos que se han introducido para formar la raíz y el primer nivel del árbol de contención. Esta regla se enuncia de una manera formal en el teorema 4.2.

Teorema 4.2

En un mecano bien formado $M = (A, C, Tb)$ no hay aristas tubo entre un asset y sus antecesores o descendientes.

$$\mathbf{BF}(M) \Rightarrow \forall (t) (\mathbf{T}(t, x, y) \wedge \mathbf{Tb}(t) \wedge \neg (\mathbf{C_t}(x, y) \vee \mathbf{C_t}(y, x))) \tag{4.5}$$

La demostración de este teorema se basa en el orden canónico para la construcción del mecano bien formado que dicta la gramática G_M. Se va a demostrar utilizando un método inductivo sobre el número de tubos bien formados del mecano $M = (A, C, Tb)$.

Caso Base

Sea el número de tubos uno. Como M está bien formado el mecano tendría dos niveles de abstracción unidos por un tubo internivel, en cualquier otro casa habrá más tubos adicionales.

Inducción

Se asume que el teorema se cumple para cualquier mecano bien formado con n tubos. Añadir un tubo $t = (x, y, i)$, con $i = 1..6$, a cualquier mecano con n tubos, debe dar como resultado otro mecano bien formado, por lo que el tubo añadido debe cumplir una de las reglas impuestas en la *definición 3.14*.

- Si el tubo t verifica **R1** (*los vértices x e y son hermanos dentro de un mismo nivel de abstracción*), es un caso trivial, ya que x no es ni antecesor ni descendiente de y.

- Si el tubo t verifica **R2** (*existe un tubo intranivel entre los padres de x e y*), por la hipótesis de inducción los padres no son descendientes el uno del otro; por tanto sus hijos tampoco.
- Si el tubo t verifica **R3** (*existe un tubo entre el padre de x e y*), por la hipótesis de inducción, el padre de x e y no pueden ser descendientes uno del otro; por tanto, x no puede ser antecesor de y, y viceversa.
- Si el tubo t verifica **R4** (*existe un tubo entre x y el padre de y*), por la hipótesis de inducción, x y el padre de y no pueden ser descendientes uno del otro; por tanto, x no puede ser descendiente de y, y viceversa.
- Si el tubo t verifica **R6-R10** (*reglas que establecen tubos internivel entre x e y*), por el mero hecho de ser tubos internivel x e y son descendientes de diferentes hijos de la raíz, no pudiendo ser en ningún caso descendientes el uno del otro.

Por tanto, este teorema se cumple para cualquier mecano bien formado.

□

4.2 Composición Automática de Mecanos en Tiempo de Reutilización

Los mecanos construidos dentro de las actividades propias del desarrollo para reutilización, se crean con el objetivo de que sean reutilizados tal cual en desarrollos con reutilización. Es decir, se entienden como configuraciones estáticas de elementos reutilizables de grano fino; lo cual significa que cada mecano tiene identidad única como elemento reutilizable dentro de la biblioteca de reutilización.

Sin embargo, si se limitase a ofrecer únicamente esta perspectiva de los mecanos, se estaría perdiendo potencia, al no poder acceder a la malla de componentes de grano fino que forman los mecanos, y flexibilidad en el proceso de reutilización, cuando los mecanos existentes no se ajusten a las necesidades de los desarrolladores con reutilización.

Para aportar esa componente de flexibilidad al proceso de reutilización se ofrece la posibilidad de formar un mecano como una configuración de assets en tiempo de reutilización. Esto significa que el mecano construido de esta forma no existía como elemento reutilizable propiamente definido dentro de la biblioteca de reutilización (*aunque puede llegar a existir como tal si se realimenta el repositorio cerrando el ciclo de reutilización*). Los mecanos con

estas características se van a construir automáticamente mediante la composición de los assets del repositorio, como resultado de una petición en un desarrollo con reutilización.

Es precisamente esta parte de construcción automática de mecanos dentro del proceso de reutilización la que lo convierte en un proceso híbrido composición/generación, entendiendo, en este caso, por generación la composición automática de un elemento reutilizable de acuerdo a proceso establecido.

4.2.1 Consideraciones a la Composición Automática de Mecanos

Todo mecano generado por composición automática, $M_g = (A_g, C_g, Tb_g)$, será un mecano bien formado, $\mathbf{BF}(M_g)$, porque su creación está dictada por la gramática dependiente del contexto G_M, cumpliéndose que:

$$M_g \in L(G_M) \tag{4.6}$$

El proceso de composición automática se sustenta en los tipos de relaciones semánticas que pueden existir entre los assets almacenados en el repositorio. Estas relaciones semánticas están soportadas por las relaciones estructurales definidas en el *Capítulo 3*.

Sin embargo, mientras que los mecanos construidos de forma "*manual*" en el proceso de desarrollo para reutilización, también denominados *mecanos persistentes*, dan lugar a elementos reutilizables perfectamente organizados, probados y cualificados que se ajustan a un determinado contexto, marcado por el modelo de proceso de reutilización [García et al., 1998d]; los mecanos "*generados*" sólo son capaces de ofrecer una aproximación de solución a una petición de reutilización que no puede ser satisfecha por los mecanos persistentes, intentando ajustarse a criterios que son fijados por el desarrollador en tiempo de reutilización.

La mayor desventaja de los mecanos generados es que ofrecen una configuración de assets que, aunque intenta aproximarse lo más posible a la solución óptima, no tiene unos criterios de cualificación y organización preestablecidos como en el caso de los mecanos persistentes. Sin embargo, la inclusión de la generación en el modelo de reutilización basado en mecanos, aporta una componente de flexibilidad que no se consigue con la utilización exclusiva de mecanos persistentes.

En un repositorio poblado con mecanos persistentes existen múltiples estructuras subyacentes de grafo inducidas por la malla de assets que los componen y por sus relaciones. Estas estructuras pueden dar lugar a mecanos que pueden componerse o extraerse en tiempo de reutilización.

De acuerdo con esto, el proceso de composición automática puede verse como la extracción de un subgrafo que satisfaga un conjunto de restricciones. Una vez que se ha obtenido el subgrafo, la formación del mecano M_g es trivial gracias a la gramática G_M. No obstante, en la formación del subgrafo se tienen tres problemas a solucionar:

1. Por dónde comenzar la construcción del subgrafo, o dicho de otra manera, por dónde encaminar la búsqueda.

2. El gran tamaño del espacio inicial de búsqueda. Este es el mayor problema de la composición automática de mecanos.

3. La más que posible mezcla de alternativas dentro del mecano generado, cuando lo deseable es que el mecano sólo presentase un camino de solución único.

Puntos de Entrada para la Composición Automática de Mecanos

Cuando un desarrollador con reutilización, ante la perspectiva de no haber encontrado mecanos persistentes en el repositorio que se ajusten a su problemática, decide componer un mecano de forma automática, debe establecer unos puntos a partir de los cuales comience el proceso de extracción automática de los componentes.

En principio estos puntos de entrada podrían ser assets de cualquier nivel de abstracción, pertenecientes a mecanos persistentes o no. Sin embargo, dejar un espacio de entrada tan abierto, a parte de repercutir en la complejidad y legibilidad del proceso en sí mismo, no es realista.

Desde la experiencia adquirida en diversos proyectos relacionados con la construcción de mecanos, parece que los puntos de entrada para este proceso automático debían de buscarse en el nivel de abstracción de requisitos, y más concretamente en aquellos assets que expresasen requisitos funcionales.

Esta decisión viene justificada porque el nivel de requisitos, además de ser el nivel de mayor abstracción, y por tanto más cercano al hombre, es donde se recoge (*se documenta*) la funcionalidad del elemento reutilizable, que será refinada en sus otros niveles de abstracción.

Además, en el Modelo de Mecano se soporta esta característica mediante los *Descriptores Funcionales*, que son elementos de clasificación que agrupan assets con descripciones funcionales en lenguaje natural[2], que sigue siendo el máximo exponente de la fase de elici-

[2]Este ha sido el criterio seguido en una primera aproximación, aunque fácilmente podía ampliarse para contar con descriptores funcionales especificados con otras técnicas.

tación de requisitos, y sobre los cuales se pueden establecer diferentes procesos de selección, ya sea vía navegación interactiva por los descriptores, o mediante técnicas de búsqueda en vocabularios controlados o en texto libre.

Esta aproximación presenta una gran restricción, debido a que únicamente aquellos mecanos que tengan asociado algún descriptor funcional podrán servir como entrada al proceso de composición automática.

Reducción del Tamaño del Espacio Inicial de Búsqueda

Un repositorio poblado con assets y mecanos puede convertirse en un espacio de búsqueda excesivamente vasto para realizar la composición automática de un nuevo mecano en tiempo de reutilización. El resultado, con casi con toda seguridad, sería un mecano de un tamaño demasiado grande para ser útil al desarrollador con reutilización, que tardaría más tiempo en comprenderlo y en desechar alternativas que en desarrollar desde cero.

Por este motivo, antes de proceder a establecer los requisitos funcionales desde los que se desea partir, se debe acotar el espacio inicial de búsqueda, suponiendo que se tiene una organización adecuada[3]. Para ello, el primer paso pasa por establecer una serie de restricciones no funcionales sobre los mecanos que servirán de base a la composición del nuevo mecano. Estas restricciones son:

- *El dominio de aplicación en el que se encuentra clasificado el mecano.*
- *Una, o varias, líneas de producto pertenecientes al dominio.*
- *El paradigma en el que se quiere obtener el mecano (Orientación a Objeto o Estructurado), para evitar mezclas sin sentido de componentes de diferentes paradigmas.*

El conjunto de mecanos que cumplan estas restricciones constituirán un espacio más reducido que el repositorio completo, y en el cual ya se pueden hacer las consultas oportunas sobre los requisitos funcionales, ya sea mediante algún procedimiento estructurado de consulta *lenguaje de consulta visual o sintáctico* o mediante navegación a través de los descriptores funcionales de los mecanos involucrados.

[3]Por otro lado imprescindible como se discute en el *Capítulo 5*.

Alternativas en el Mecano Generado

Como oportunamente se indicó al estudiar el Modelo de Mecano, dentro de un mecano *persistente* no hay alternativas, es decir, todas los enlaces entre sus assets componentes son necesarios para cumplir los objetivos del mecano. Sin embargo, cuando se construye un mecano por composición automática se combinan varios mecanos en los que es más que probable que se presenten caminos alternativos, que se vean reunidos en el elemento reutilizable resultado de este proceso.

En este sentido, el proceso de composición automática no es capaz de elegir entre las alternativas posibles, aunque sí permite identificar que caminos son alternativos, para que un tratamiento posterior, ya sea automático o manual, permita elegir al desarrollador con reutilización que alternativa elegir.

Los tipos de relaciones estructurales que pueden dar lugar a alternativas, y por tanto conflictos, son:

- **Reificación:** Es una relación conflictiva bajo la perspectiva del proceso de composición automática de mecanos porque un asset clasificado en un nivel de abstracción n puede reificarse en uno o varios assets de nivel de abstracción $n+1$, dentro del mismo esfuerzo de desarrollo. Lo cual impide distinguir estas situaciones de la introducción de alternativas en la reificación de dicho asset.

- **Usa a:** Es una relación conflictiva bajo la perspectiva del proceso de composición de mecanos porque un determinado asset puede ser cliente de varios assets servidores, estableciéndose varias relaciones de dependencia dentro de un mismo esfuerzo de desarrollo. Este hecho impide distinguir la situación en la que un asset cliente disponga de diferentes alternativas para obtener un mismo servicio.

- **Extensión:** El problema de la relación de extensión está relacionado con su enlace fuerte (*que se orienta del asset derivado al asset base*) en las situaciones como las que refleja la figura 4.5, donde **d1** puede ser una extensión de **p1** y de **p2** dentro de un mismo mecano, o bien puede ser una extensión de **p1** en un mecano o **p2** en otro.

- **Asociación fuerte:** Es una relación conflictiva con respecto al proceso de composición automática de mecanos, porque un asset puede estar asociado a varios assets dentro de un mismo mecano, pero puede tener asociaciones a assets alternativos en otros mecanos.

Ni la agregación y ni la composición son relaciones conflictivas para el proceso de

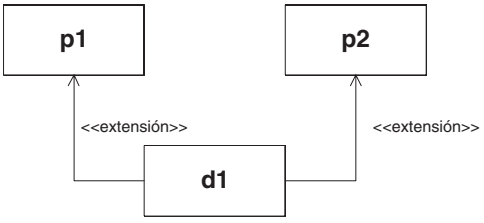

Figura 4.5: Situación conflictiva con respecto a la relación de extensión.

composición automática de mecanos porque en la agregación/composición de un conjunto de assets para formar un asset agregado/compuesto no puede haber alternativas, por lo que todos los assets que forman el agregado/compuesto pertenecen al mismo esfuerzo de desarrollo.

La asociación débil no es una relación conflictiva bajo la perspectiva del proceso de composición automática de mecanos porque los dos enlaces dirigidos en direcciones opuestas que de ella se derivan son débiles, y por lo tanto no se tendrán en cuenta por dicho proceso.

4.2.2 Proceso de Composición Automática de Mecanos

El proceso de composición automática de mecanos tiene como cometido la obtención de un elemento reutilizable de grano grueso que se ajuste lo más posible a una solicitud de un desarrollador con reutilización sobre los descriptores funcionales de los mecanos *persistentes* almacenados en el repositorio.

Es importante puntualizar que el proceso de composición automática que se presenta a continuación se ajusta a un conjunto de restricciones, que evitan que el centro de atención del proceso se vea desplazado por intentar contemplar todas las variantes que se pueden dar en una implementación real. Así, el proceso expuesto toma como base *mecanos persistentes* que cuentan con algún *descriptor funcional*, y se fija el nivel de entrada, para el proceso de extracción de los assets, el nivel de requisitos, más concretamente las descripciones de requisitos funcionales.

El esquema general del proceso de composición automática de mecanos es:

1. *Establecimiento del contexto del mecano a generar.*
2. *Selección de los descriptores de los requisitos funcionales.*
3. *Creación del árbol de contención del nuevo mecano.*

4. Establecimiento de las aristas tubo del mecano.

Establecimiento del Contexto del Mecano a Generar

Con el objetivo de reducir el espacio inicial de búsqueda, se fijan los valores de un dominio, de una (*o varias*) línea de producto dentro del dominio y del paradigma en el que se va a trabajar (*OO o Estructurado*).

Sólo los mecanos *persistentes* que tengan algún descriptor funcional asociado serán los que formen el espacio inicial donde buscar los requisitos funcionales del mecano a generar.

Se denota por $M_{inicial}$ al conjunto de mecanos seleccionados tras quedar establecido el contexto.

Selección de los Descriptores de los Requisitos Funcionales

Realizando algún tipo de consulta, o navegando a través de los descriptores funcionales de los mecanos que forman el espacio inicial de búsqueda, el desarrollador con reutilización selecciona el conjunto de requisitos que debe intentar satisfacer el mecano generado.

Este proceso debe refinarse lo máximo posible hasta obtener el conjunto de requisitos mínimo a satisfacer. El conjunto de descriptores de requisitos funcionales seleccionado formará el descriptor funcional del mecano generado.

Se denota por R al conjunto de requisitos funcionales que han sido seleccionados en esta fase.

$$R = \{r_1, r_2, \ldots r_n\} \tag{4.7}$$

Se denota por M_p al conjunto de mecanos que están en $M_{inicial}$ y tienen como componente a alguno de los requisitos seleccionados.

$$M_p = \{M_i(A_i, C_i, Tb_i) \in M_{inicial} \mid R \cap A_i \neq \emptyset\} \tag{4.8}$$

M_p va a servir como entrada para el proceso de extracción automática de los assets que van a ser los componentes del mecano generado.

Se denota por R_{M_i} a aquellos requisitos seleccionados que pertenecen al mecano $M_i \in M_p$.

$$R_{M_i} = R \cap A_i \neq \emptyset \tag{4.9}$$

Creación del Árbol de Contención del Nuevo Mecano

Aquí es donde comienza el proceso de composición automática propiamente dicho. Se parte de un conjunto de descriptores de requisitos funcionales, R, y de un conjunto de mecanos persistentes, M_p, obteniéndose el árbol de contención (A_g, C_g) del mecano M_g generado mediante composición automática.

Los pasos a dar para la construcción del árbol de contención (A_g, C_g) de M_g son:

1. Aplicar la regla *P0d* de G_M para construir la raíz y el primer nivel de (A_g, C_g)

2. Para cada $M_i \in M_p$, con $i = 1.. \mid M_p \mid$

2.1 Se insertan los descendientes de n_1 (*nivel de requisitos*)

2.1.1 Para cada $r_j \in R_{M_i}$, con $j = 1.. \mid R_{M_i} \mid$

2.1.1.1 r_j se inserta en $A_{c_1 i}$, ($A_{c_1 i}$ = {Assets candidatos de M_i para n_1 de M_g}).

2.1.1.2 Se siguen recursivamente sus enlaces intranivel con con carácter fuerte, insertando todos los destinos de estos enlaces en el conjunto $A_{c_1 i}$,

2.1.2 Para cada $a_k \in A_{c_1 i}$

2.1.2.1 Si $a_k \notin A_g$ entonces a_k se introduce en A_g, y en (A_g, C_g) como descendiente de n_1 utilizando la regla oportuna de G_M: *P1a, P2a* ó *P2b*, dependiendo de la profundidad de a_k en M_i

2.2 Se insertan los descendientes de n_2 (*nivel de diseño*)

2.2.1 Para cada $a_k \in A_{c_1 i}$

2.2.1.1 Se siguen sus enlaces internivel con destino en un asset de diseño, insertando los assets destinos de estos enlaces en el conjunto $A_{c_2 i}$, donde $A_{c_2 i}$ = {*Assets candidatos de M_i para el nivel n_2 de M_g*}

2.2.2 Para cada $a_k \in A_{c_2 i}$

2.2.2.1 Se siguen recursivamente sus enlaces intranivel con caracter fuerte, insertando los assets destinos de estos enlaces en el conjunto $A_{c_2 i}$

2.2.3 Para cada $a_k \in A_{c_2 i}$

2.2.3.1 Si $a_k \notin A_g$ entonces a_k se introduce en A_g, y en (A_g, C_g) como descendiente de n_2 utilizando la regla oportuna de G_M: *P1a, P2a* ó *P2b*, dependiendo de la profundidad de a_k en M_i

2.3 Se insertan los descendientes de n_3 (*nivel de implementación*)

2.3.1 Para cada $a_k \in A_{c_1 i}$

2.3.1.1 Se siguen sus enlaces internivel con destino en un asset de implementación, insertando los assets destinos de estos enlaces en el conjunto $A_{c_3 i}$, donde $A_{c_3 i} = \{Assets\ candidatos\ de\ M_i\ para\ el\ nivel\ n_3\ de\ M_g\}$

2.3.2 Para cada $a_k \in A_{c_2 i}$

2.3.2.1 Se siguen sus enlaces internivel con destino en un asset de implementación, insertando los assets destinos de estos enlaces en el conjunto $A_{c_3 i}$

2.3.3 Para cada $a_k \in A_{c_3 i}$

2.3.3.1 Se siguen recursivamente sus enlaces intranivel con caracter fuerte, insertando los assets destinos de estos enlaces en el conjunto $A_{c_3 i}$

2.3.4 Para cada $a_k \in A_{c_3 i}$

2.3.4.1 Si $a_k \notin A_g$ entonces a_k se introduce en A_g, y en (A_g, C_g) como descendiente de n_3 utilizando la regla oportuna de G_M: *P1a, P2a* ó *P2b*, dependiendo de la profundidad de a_k en M_i

2.4 $AssetsUsadosM_i = A_{c_1 i} \cup A_{c_2 i} \cup A_{c_3 i}$

3. Si $d(n_j, C) = 0$, con $j = 2..3$ entonces eliminar n_j

Establecimiento de las Aristas Tubo del Mecano

Una vez que se tiene creado el árbol de contención del mecano a generar, M_g, el siguiente paso es establecer las aristas tubo entre los assets. Como paso previo se debe aplicar la regla de producción *P2c* para convertir las aristas U en aristas R.

Para tener constancia de que mecano es el responsable de la inserción de un tubo, y por tanto de la existencia de alternativas, se va a denotar por t_{ij} al tubo de tipo i proveniente del mecano M_j, con $i = 1..5$ y $j = 1..\mid M_p \mid$.

Para insertar los tubos se siguen los siguientes pasos:

1. Para cada $M_j \in M_p$, con $j = 1..\mid M_p \mid$

1.1 Para cada $a_k \in AssetsUsadosM_j$ y $profunfidad(a_k) \geq 2$ se crean sus tubos intranivel

1.1.1 Para cada $a_l \in AssetsUsadosM_j$, con a_l clasificado en el mismo nivel de abstracción que a_k y $profundidad(a_l) - profundidad(a_k)$ es mínima

1.1.1.1 Si existe un tubo $(a_k, a_l, i) \in M_j$, con $i = 2..5$, se crea el tubo $t_{ij} = (a_k, a_l, i)$ en M_g aplicando la regla de producción oportuna *P4-P7*;

siempre y cuando no se haya creado previamente, es decir no exista ya un $t_{im} = (a_k, a_l, i) \mid t_{im} \in Tb_g$, con $1 \leq m \leq j-1$ y $j > 1$

1.2 Para cada $a_k \in AssetsUsadosM_j$ tal que $n_1\, C^+ a_k$ ó $n_2\, C^+ a_k$ y $profunfidad(a_k) \geq 2$, se crean sus tubos internivel

1.2.1 Para cada $a_l \in AssetsUsadosM_j$, con a_l clasificado en un nivel de menor abstracción que a_k, y tal que $profundidad(a_l) - profundidad(a_k)$ es mínima

1.2.1.1 Si existe un tubo $(a_k, a_l, 1) \in M_j$, se crea el tubo $t_{1j} = (a_k, a_l, 1)$ en M_g aplicando la regla de producción oportuna *P8-P10*; siempre y cuando no se haya creado previamente, es decir no exista ya un $t_{1m} = (a_k, a_l, 1) \mid t_{im} \in Tb_g$, con $1 \leq m \leq j-1$ y $j > 1$

Una vez que el mecano ha sido generado, se promocionan todas sus aristas de contención a aristas finales, aplicando las reglas de producción *P1b* y *P3* según el caso.

4.2.3 Tratamiento de las Alternativas en el Mecano Generado

En el mecano generado, $M_g(A_g, C_g, Tb_g)$, se establecen diferentes caminos alternativos desde los requisitos a la implementación, estas alternativas podían haberse reducido fijando alguna restricción más relacionada con el lenguaje de programación o la plataforma de uso de los assets, aunque estas y otras opciones serán responsabilidad del entorno operativo para la reutilización ofrecido por la biblioteca de reutilización que acoja el modelo de reutilización propuesto.

Para cada asset $a_k \in A_g$ con respecto a cada tipo de tubo i, con $i = 1..5$, se cumple que:

$$d(a_k, i) = \sum_{j=1}^{|M_p|} d(a_k, t_{ij}) \tag{4.10}$$

Un asset a_k no tiene alternativas con respecto a un tipo de tubo i, con $i = 1..5$, si se cumple que:

$$\exists\, j, \text{ con } j = 1.. \mid M_p \mid \setminus\, d(a_k, i) = d(a_k, t_{kj}) \tag{4.11}$$

Un asset a_i tiene tantas alternativas con respecto al tipo de tubo t_k como:

$$\mid d(a_i, t_{kj}) > 0 \mid \tag{4.12}$$

El tratamiento de las alternativas puede hacer mediante técnicas manuales y/o automáticas, siendo servicios que debe aportar la biblioteca de reutilización. A continuación se comentan algunas aproximaciones al tratamiento de las alternativas.

- *El desarrollador con reutilización decide de forma interactiva:* Al desarrollador con reutilización se le presenta M_g destacando las diferentes alternativas para que el vaya juzgando cuales desecha. Este servicio debe cuidar que cada vez que el desarrollador con reutilización elimine una alternativa, el mecano resultando siga estando bien formado.

- *Descomposición en mecanos:* Se le ofrece al desarrollador con reutilización un mecano por cada alternativa. En el caso de haber muchas puede llegar a ser una carga para el desarrollador con reutilización.

- *Ajuste según un patrón:* Se pueden contar con patrones que se ajusten a diferentes enfoques metodológicos, de forma que de las diferentes alternativas se seleccione aquella (*o aquellas*) que mejor se ajuste al patrón.

- *Selección de la mejor solución o búsqueda hacia adelante:* Se intenta buscar la mejor solución descartando todas las alternativas de cada asset menos una. Para, descartar alternativas en un determinado nivel se busca información en los niveles inferiores. Si aún así, siguen existiendo alternativas, se debe recurrir a diversos heurísticos (*al estilo de las "recetas" para desarrollar de forma automática utilizando frameworks propuestas en [Schappert et al., 1995]*), tales como potenciar los mecanos más completos (*sin falta de niveles*), potenciar mecanos con componentes de mayor cualificación, potenciar mecanos con componentes ampliamente reutilizados y probados...

4.2.4 Ejemplo de Composición Automática de un Mecano

Para la mejor comprensión del proceso de composición automática de un mecano, se va a presentar un ejemplo sencillo.

Se va a suponer que tras la selección del dominio, línea de producto y paradigma el espacio inicial de búsqueda está formada por los mecanos $Mecano_1$ (*figura 4.6*) y $Mecano_2$ (*figura 4.7*); $M_p = \{Mecano_1, Mecano_2\}$.

El descriptor funcional del $Mecano_1$ está formado por los descriptores de requisitos funcionales *RF1, RF2, RF3* y *RF4*, mientras que el descriptor funcional del $Mecano_2$ está formado por los descriptores de requisitos funcionales *RF1, RF2* y *RF5*.

Suponiendo que el desarrollador con reutilización selecciona que se genere por composición automática un mecano M_g que tenga como requisitos funcionales *RF1* y *RF5*, $R = \{RF1, RF5\}$, el proceso a seguir sería el siguiente.

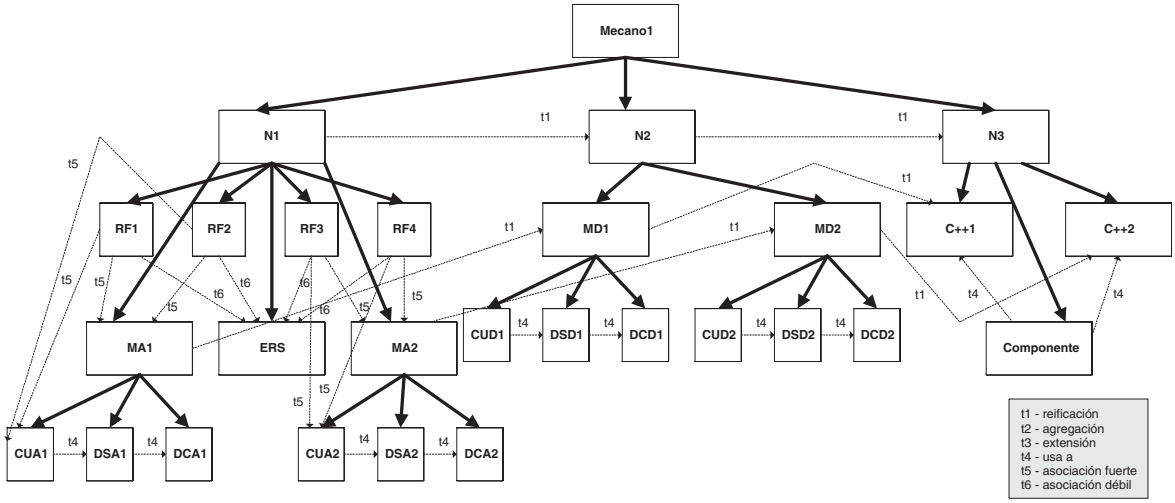

Figura 4.6: $Mecano_1$.

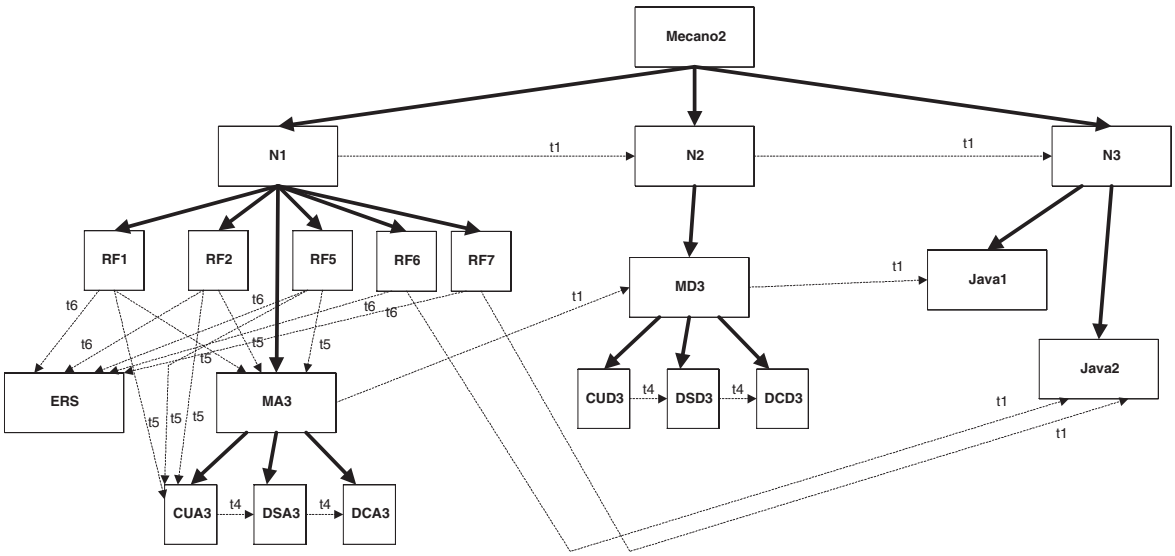

Figura 4.7: $Mecano_2$.

1. *Se crea la raíz y el primer nivel de M_g, aplicando la regla P0d.*

2. *Se construye el nivel de abstracción n_1 de M_g con respecto a $Mecano_1$. Para ello hay que, partiendo de los descriptores de requisitos funcionales elegidos, $R_{Mecano_1} = \{RF1\}$, encontrar los assets clasificados en N_1 que son destinos de enlaces intranivel fuertes. Así, para $Mecano_1$, $A_{c_1} = \{RF1, MA1, CUA1, DSA1, DCA1, RF2\}$*

3. *Como ninguno de los elementos de A_{c_1} pertenece a A_g, se insertan en M_g aplicando la regla P1a para introducir RF1, P2a para introducir MA1 y RF2, P2b para introducir CUA1 y P2a para introducir DSA1 y DCA1. Así, el árbol de contención de M_g después de este paso se refleja en la figura 4.8.*

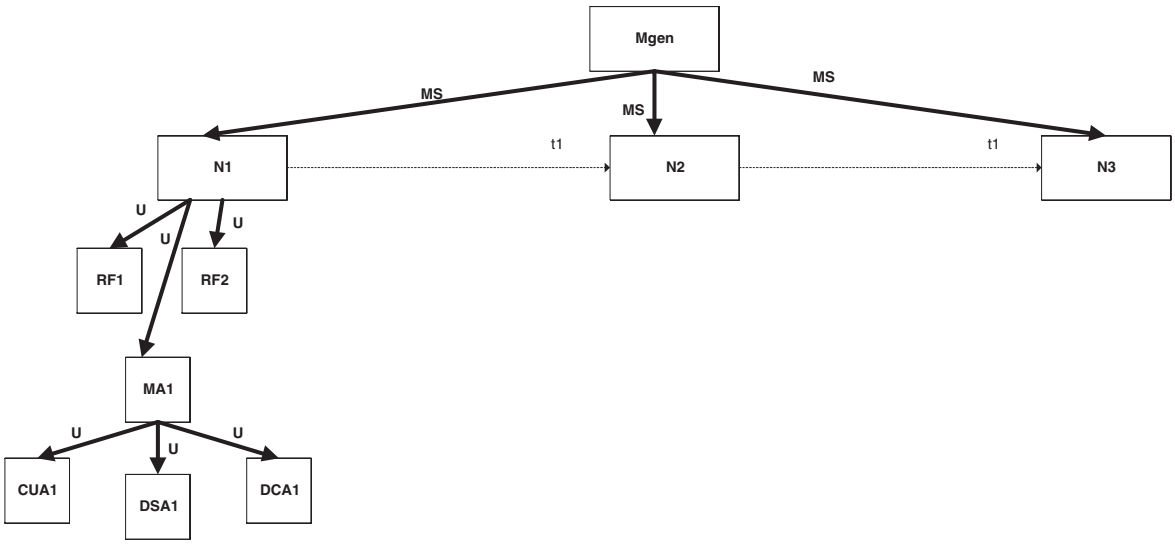

Figura 4.8: Árbol de contención parcial de M_g.

4. *Se construye el nivel de abstracción n_2 de M_g con respecto a $Mecano_1$. Para hacerlo, se buscan los enlaces internivel de los elementos de A_{c_1} hacia el nivel N_2, encontrando sólo al elemento MD1; partiendo de MD1 se siguen sus enlaces intranivel fuertes y se obtienen CUD1, DSD1 y DCD1, con lo que $A_{c_2} = \{MD1, CUD1, DSD1, DCD1\}$.*

5. *Como ninguno de los elementos de A_{c_2} está en A_g se introducen en M_g aplicando las reglas P1a para introducir MD1, P2b para introducir CUD1, y P2a para hacer lo propio con DSD1 y DCD1.*

6. *Se construye el nivel de abstracción n_3 de M_g con respecto a $Mecano_1$. Se sigue para ello un proceso similar a los anteriores niveles, primero se buscan los enlaces internivel de los elementos de A_{c_1} hacia el nivel n_3, en este caso no hay ninguno; luego los enlaces internivel de los elementos de A_{c_2} hacia el nivel n_3, encontrando sólo al elemento C++1, elemento que no tiene ningún enlace intranivel fuerte, entonces $A_{c_3} = \{C++1\}$.*

7. *Como el elemento de A_{c_3} no está en A_g, se introduce aplicando la regla P1a.*

8. *El árbol de contención de M_g con respecto a $Mecano_1$ está terminado, como se muestra en la figura 4.9, y además se tiene que:*

$$AssetsUsadosMecano_1 = \{RF1,\ RF2,\ MA1,\ CUA1,\ DSA1,\ DCA1,\ MD1,\ CUD1,\ DSD1,\ DCD1,\ C++1\}$$

$$A_g = \{M_{gen},\ N1,\ N2,\ N3,\ RF1,\ RF2,\ MA1,\ CUA1,\ DSA1,\ DCA1,\ MD1,\ CUD1,\ DSD1,\ DCD1,\ C++1\}$$

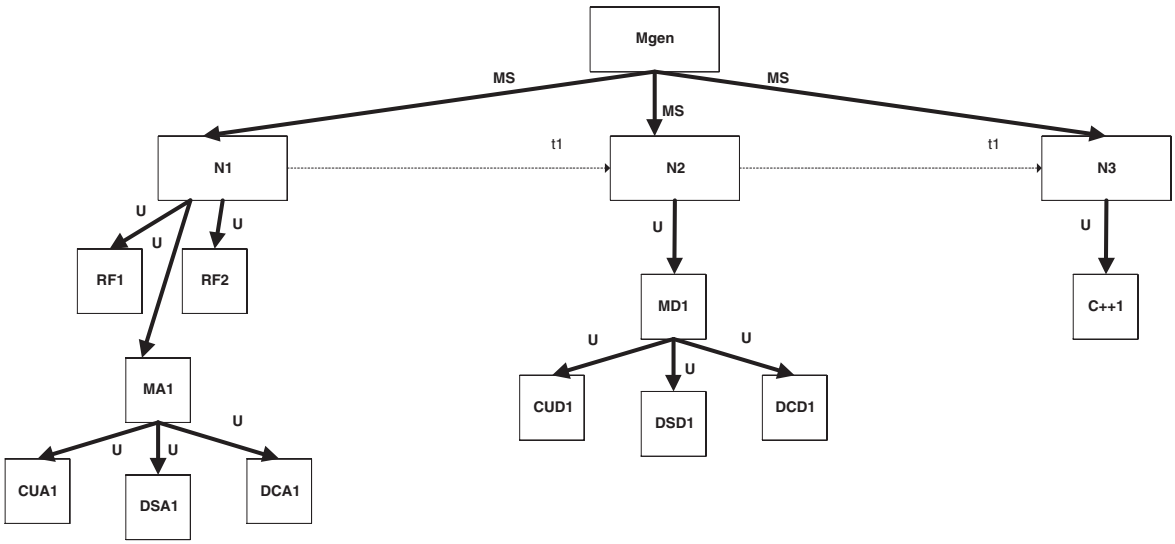

Figura 4.9: Árbol de contención parcial de M_g con respecto a $Mecano_1$.

9. *Se construye el nivel de abstracción n_1 de M_g con respecto a $Mecano_2$. Para lo cual se van a seguir los mismos pasos que para $Mecano_1$, esto es, partiendo de los descriptores de requisitos funcionales $R_{Mecano_2} = \{RF1,\ RF5\}$ se siguen sus enlaces intranivel fuertes, tal que $A_{c_1} = \{RF1,\ RF5,\ MA3,\ CUA3,\ DSA3,\ DCA3,\ RF2\}$, para $Mecano_2$.*

10. *Todos los elementos de A_{c_1} se insertan en M_g a excepción de RF1 y RF2 que ya están en A_g. Para esta operación se recurre a la regla P2a para incorporar a RF5 y a MA3 y a la regla P2b para incorporar el primer hijo de MA3, esto es, CUA3, de nuevo a la regla P2a para añadir a DSA3 y a DCA3.*

11. *Se construye el nivel de abstracción n_2 de M_g con respecto a $Mecano_2$. Siguiendo los enlaces internivel de los elementos de A_{c_1}, se obtiene MD3, y a partir de sus enlaces intranivel se llega a CUD3, DSD3 y DCD3;* $A_{c_2} = \{MD3, CAUD3, DSD3, DCD3\}$.

12. *Como ninguno de los elementos de A_{c_2} está en A_g se introducen en M_g aplicando las reglas P2a para introducir MD3, P2b para introducir CUD3, y P2a para hacer lo propio con DSD3 y DCD3.*

13. *Se construye el nivel de abstracción n_3 de M_g con respecto a $Mecano_2$. Primeramente se buscan los enlaces internivel de los elementos de A_{c_1} hacia el nivel n_3, en este caso no hay ninguno; luego los enlaces internivel de los elementos de A_{c_2} hacia el nivel n_3, encontrando sólo al elemento Java1, elemento que no tiene ningún enlace intranivel fuerte, entonces* $A_{c_3} = \{Java1\}$.

14. *Como el elemento de A_{c_3} no está en A_g, se introduce aplicando la regla P2a.*

15. *El árbol de contención de M_g está terminado, como se muestra en la figura 4.10, y además se tiene que:*

$$AssetsUsadosMecano_2 = \{RF1, RF2, RF5, MA3, CUA3, DSA3, DCA3, MD3, CUD3, DSD3, DCD3, Java1\}$$

$$A_g = \{M_{gen}, N1, N2, N3, RF1, RF2, RF5, MA1, CUA1, DSA1, DCA1, MA3, CUA3, DSA3, DCA3, MD1, CUD1, DSD1, DCD1, MD3, CUD3, DSD3, DCD3, Java1\}$$

16. *Se convierten todas las aristas U a aristas R.*

17. *Se insertan las aristas tubo intra e internivel en M_g con respecto a $Mecano_1$. Como en este momento $T_g = \emptyset$, todas las aristas tubo se insertan en M_g utilizando las reglas de producción que se necesiten entre P4-P10. Tras esta operación M_g ofrece la vista que se muestra en la figura 4.11.*

18. *Se insertan las aristas tubo intra e internivel en M_g con respecto a $Mecano_2$. Tras esta operación M_g ya tiene todas sus aristas de contención y tubo, y el resultado se presenta en la figura 4.12. Es interesante llamar la atención sobre las alternativas que presenta este mecano generado.*

19. *Para terminar, se convierten todas las aristas de contención en aristas terminales, utilizando las reglas P1b y P2d.*

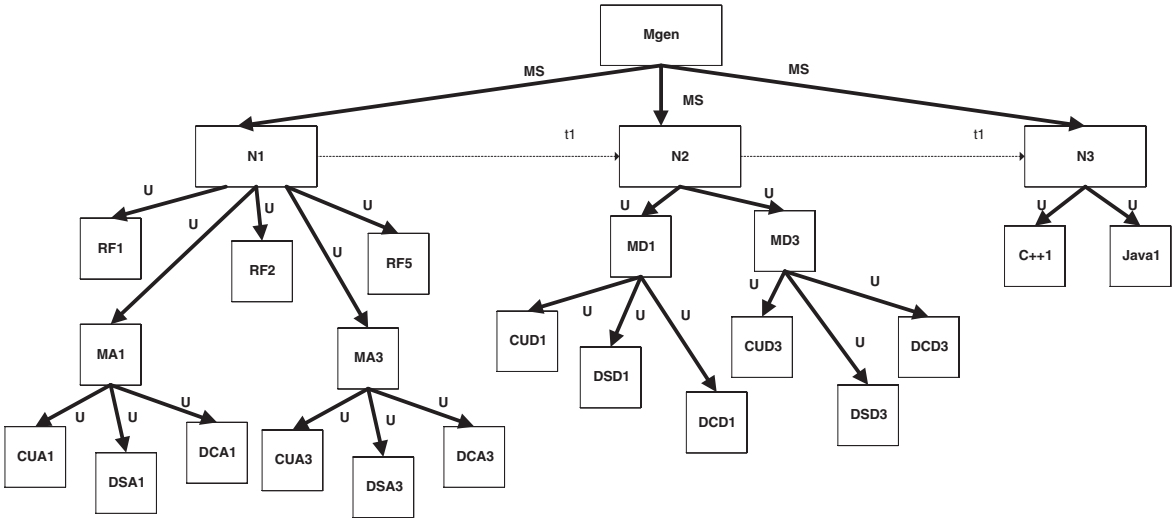

Figura 4.10: Árbol de contención de M_g.

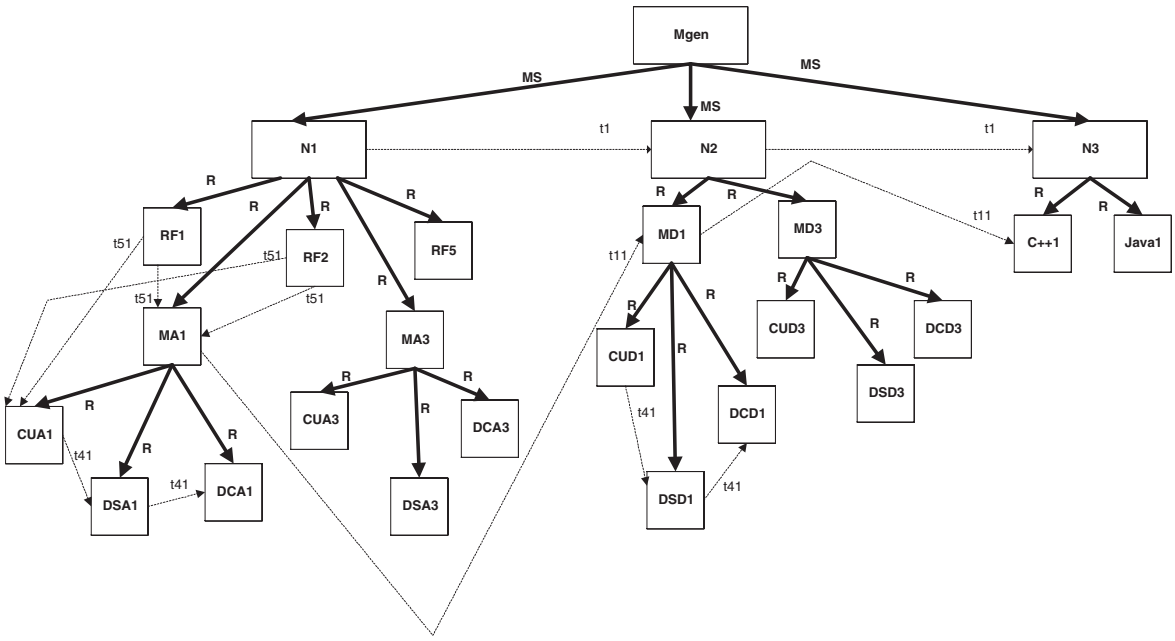

Figura 4.11: Vista de M_g tras la inserción de los tubos de $Mecano_1$.

4.2.5 Conclusiones sobre la Composición Automática de Mecanos

Con este proceso de creación de mecanos complementario al propio del desarrollo para reutilización se introduce un grado de flexibilidad mayor en el proceso de reutilización.

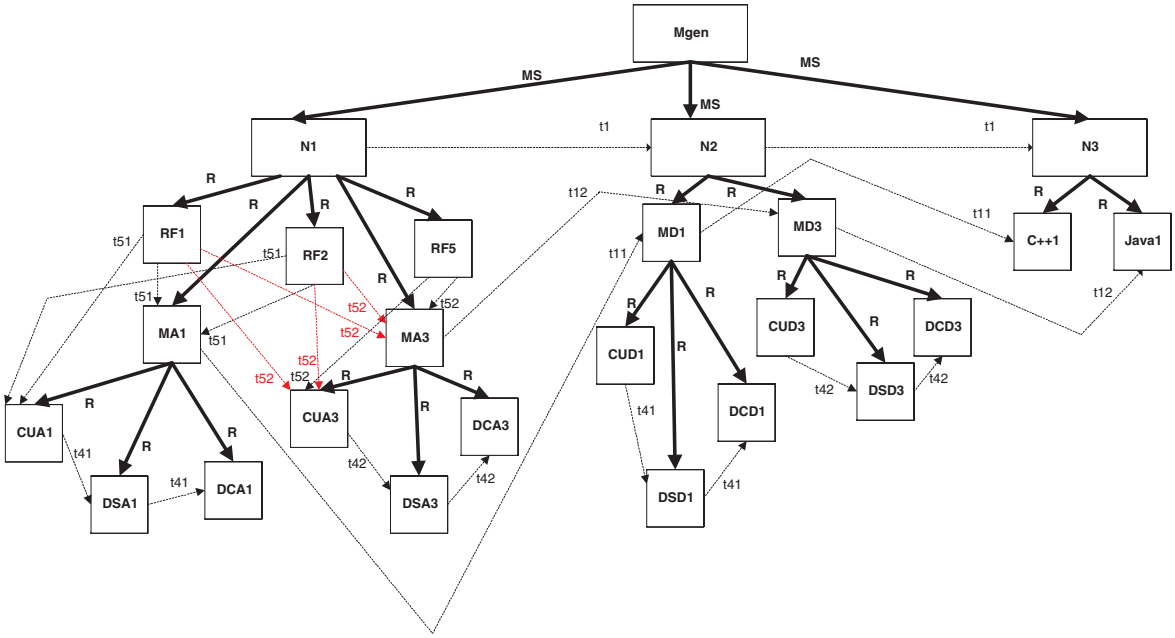

Figura 4.12: Mecano generado por composición automática M_g.

Es importante recalcar el hecho de que el grano de reutilización que se está manejando en el modelo de reutilización propuesto es el marcado por los distintos productos obtenidos a lo largo del ciclo de vida de un desarrollo software. En estas condiciones el mecano generado difícilmente ofrecerá la información exacta que el desarrollador con reutilización solicitó, ofreciendo una solución que se aproximará por exceso o por defecto a lo pedido, y en la que posible e inevitablemente podrán aparecer elementos solapados.

Además, se debe tener presente que un mecano generado no podrá tener la certificación y cualificación con la que puede contar un mecano persistente, cuyos componentes han sido elegidos para formar un elemento reutilizable de acuerdo a un proceso de desarrollo para reutilización, que ha sido posteriormente auditado y certificado.

El núcleo del proceso de composición automática está basado en la gramática G_M, por lo que siempre dará como resultado un mecano bien formado.

Capítulo 5

Modelo de Reutilización Basado en Mecanos

La introducción de la reutilización sistemática dentro de un organismo requiere el establecimiento de un modelo que cubra todos los aspectos del área, y se integre en las actividades de organización y de desarrollo del propio organismo. Así pues, un modelo de reutilización se convierte en un síntoma inequívoco de madurez dentro de la política de desarrollo de la organización.

Este modelo es el responsable de la canalización de todas las situaciones derivadas de la creación, organización y evaluación de los elementos reutilizables que pertenecen al organismo.

Existen diversos modelos de reutilización recogidos en la bibliografía especializada; algunos de los más destacados se presentan someramente a continuación.

El modelo de reutilización propuesto por Prieto-Díaz y Freeman [Prieto-Díaz and Freeman, 1987b], que está centrado en la clasificación de los elementos reutilizables[1] (*usando el esquema de facetas definido por el propio Prieto-Díaz [Prieto-Díaz, 1989], [Prieto-Díaz, 1991]*) para facilitar su pronta recuperación por parte de un desarrollador con reutilización y la selección del elemento más adecuado cuando existan varios candidatos. Roger S. Pressman presenta en [Pressman, 1997] un modelo de reutilización para, con fines docentes, introducir la reutilización del software; este modelo, aunque centrado en los apartados más técnicos del proceso de desarrollo para y con reutilización, recoge aspectos relacionados con la clasificación de los elementos reutilizables y los factores económicos de la reutilización

[1] Se centra en assets de código fuente.

del software. David C. Rine y Nader Nada han desarrollado el *Software Reuse Manufacturing Reference Model* (***RRM*** - *Reuse Reference Model*) [Rine and Nada, 1998] como una solución práctica para disminuir el esfuerzo de desarrollo del software, aumentando la calidad y disminuyendo el tiempo de mercado del mismo, a través de la reutilización; en este proceso se distinguen dos partes principales: la vertiente técnica y la vertiente de proceso. En [Biddle and Tempero, 1998] se presenta un modelo de reutilización basado en unidades del nivel implementación que pueden ser ensambladas, al estilo de un modelo de componentes.

En esta propuesta de tesis doctoral se presenta el marco de referencia de un modelo de reutilización definido sobre la base de los mecanos. Este modelo de reutilización, al que a partir de ahora se hará referencia como **MRG** (*Modelo de Reutilización GIRO*), intenta recoger las tareas a realizar en cada uno de los apartados que guían la construcción y utilización de los elementos reutilizables de grano grueso denominados mecanos. Más concretamente, el MRG consta de tres partes o submodelos de reutilización; el primero de ellos se encarga de las técnicas y herramientas empleadas para la construcción de los mecanos, el segundo de ellos tiene como cometido la parte de proceso de reutilización, es decir, cómo se organiza el enfoque de reutilización, y por último el tercero se ocupa del apartado de la cualificación de los mecanos como medio para certificar su calidad.

El capítulo se organiza en cuatro secciones. La primera sección realiza una presentación general del modelo de reutilización MRG, explicando sus características más relevantes. La sección dos está dedicada al modelo técnico de reutilización, haciendo de nuevo hincapié en el enfoque híbrido composición/generación que permiten los mecanos. La sección tercera sirve para perfilar el modelo de proceso de reutilización, de forma que el núcleo del mismo está basado en la relación de los mecanos con el enfoque de mayor auge de la reutilización sistemática del software, las líneas de producto. En la cuarta y última sección se presenta el modelo de cualificación de reutilización, definiendo el marco general del mismo, en el apartado más abierto de todo el modelo MRG.

5.1 Introducción al Modelo de Reutilización MRG

Los mecanos son el elemento unificador que se utiliza para construir el modelo de reutilización MRG. Este modelo de reutilización presenta tres vistas diferentes, aunque con importantes interdependencias: *el modelo técnico de reutilización, el modelo de proceso de reutilización* y *el modelo de cualificación de reutilización* [García et al., 1998d], tal y como se refleja en la figura 5.1.

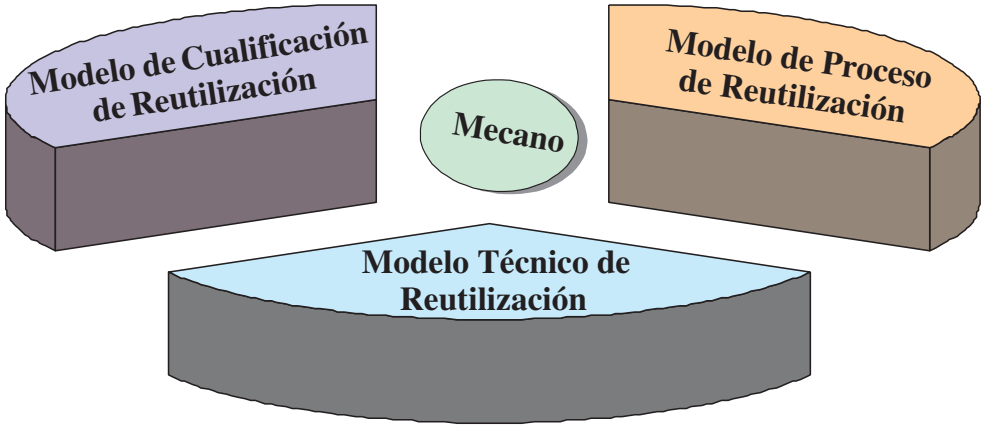

Figura 5.1: Vistas en el modelo de reutilización MRG.

Aunque cada una de estas vistas va a ser objeto de un estudio más detallado en las siguientes secciones de este capítulo, a modo de resumen introductorio, se puede decir que:

- El modelo técnico de reutilización es el conjunto de conceptos y actividades que se utilizan para construir los elementos software reutilizables.
- El modelo de proceso de reutilización centra su atención en la organización lógica del personal, recursos, métodos y procedimientos, para definir la política y la gestión de la reutilización dentro de un organismo.
- El modelo de cualificación para la reutilización hace referencia a los criterios de certificación de los mecanos, criterios que se van a tomar en función de un proceso de auditoría y de unas métricas, que serán resultado de un plan de calidad fijado.

En general, el modelo de reutilización MRG en una aproximación global se puede decir que se caracteriza por los siguientes aspectos:

- *Basado en los mecanos:* El centro sobre el que gira todo el MRG es el mecano, que en definitiva es el elemento reutilizable que se maneja, aunque éste sea una configuración de elementos de menor grano.
- *Aborda todo el proceso general de reutilización:* Las actividades de desarrollo para reutilización y de desarrollo con reutilización marcan el proceso de reutilización del modelo MRG.

- *Aproximación híbrida de reutilización:* Las dos perspectivas soportadas en la construcción de los mecanos, construcción por composición y generación por composición automática, le confiere al MRG ese carácter mixto composición/generación, presentando una mayor capacidad y flexibilidad para obtener elementos reutilizables en el desarrollo con reutilización.

- *Soportado por herramientas:* El modelo MRG requiere de una biblioteca de reutilización que le sirva como soporte operativo para su proceso de reutilización, con todos los servicios que requiere para sus diferentes facetas.

- *Organizado en líneas de producto:* Las líneas de producto presentan la aproximación con la se obtienen los mejores resultados dentro de un enfoque sistemático de reutilización, siendo la organización dictada por el MRG, y a la que los mecanos se adaptan de una forma natural, como se puede desprender del propio Modelo de Mecano.

- *Soporte para la calidad en el desarrollo de los productos software:* Uno de los objetivos más importantes de la reutilización del software es tener la certificación de que los bloques software que se utilizarán en el desarrollo con reutilización tienen garantizada un nivel de calidad y confieren seguridad a los productos desarrollados a partir de ellos. Esto se tiene presente en el MRG, definiendo un proceso de cualificación de elementos reutilizables que garantice la auditoría y verificación de los mismos, bajo el auspicio de un plan de calidad adecuado.

5.2 Modelo Técnico de Reutilización

El modelo técnico de reutilización del MRG se centra en todos los elementos tecnológicos, herramientas y actividades, necesarios para la creación de mecanos. Es decir, define el modelo de componente reutilizable a utilizar (*mecanos como agregado de assets clasificados en diferentes niveles de abstracción e interrelacionados entre sí*) y el modelo de repositorio para almacenar todos los datos relevantes y necesarios para ofrecer el soporte de reutilización basada en mecanos.

Los aspectos técnicos del modelo de reutilización que se está definiendo han quedado expuestos en los *capítulos 3 y 4*, donde se ha tratado el modelo de componente reutilizable desde una doble perspectiva (*semi-formal y formal*), y el modelo de respositorio, desde una perspectiva semi-formal.

En la figura 5.2 se presenta el esquema de la arquitectura de entorno operativo para la reutilización, que sobre el modelo de repositorio y el modelo de mecano, define el modelo

técnico.

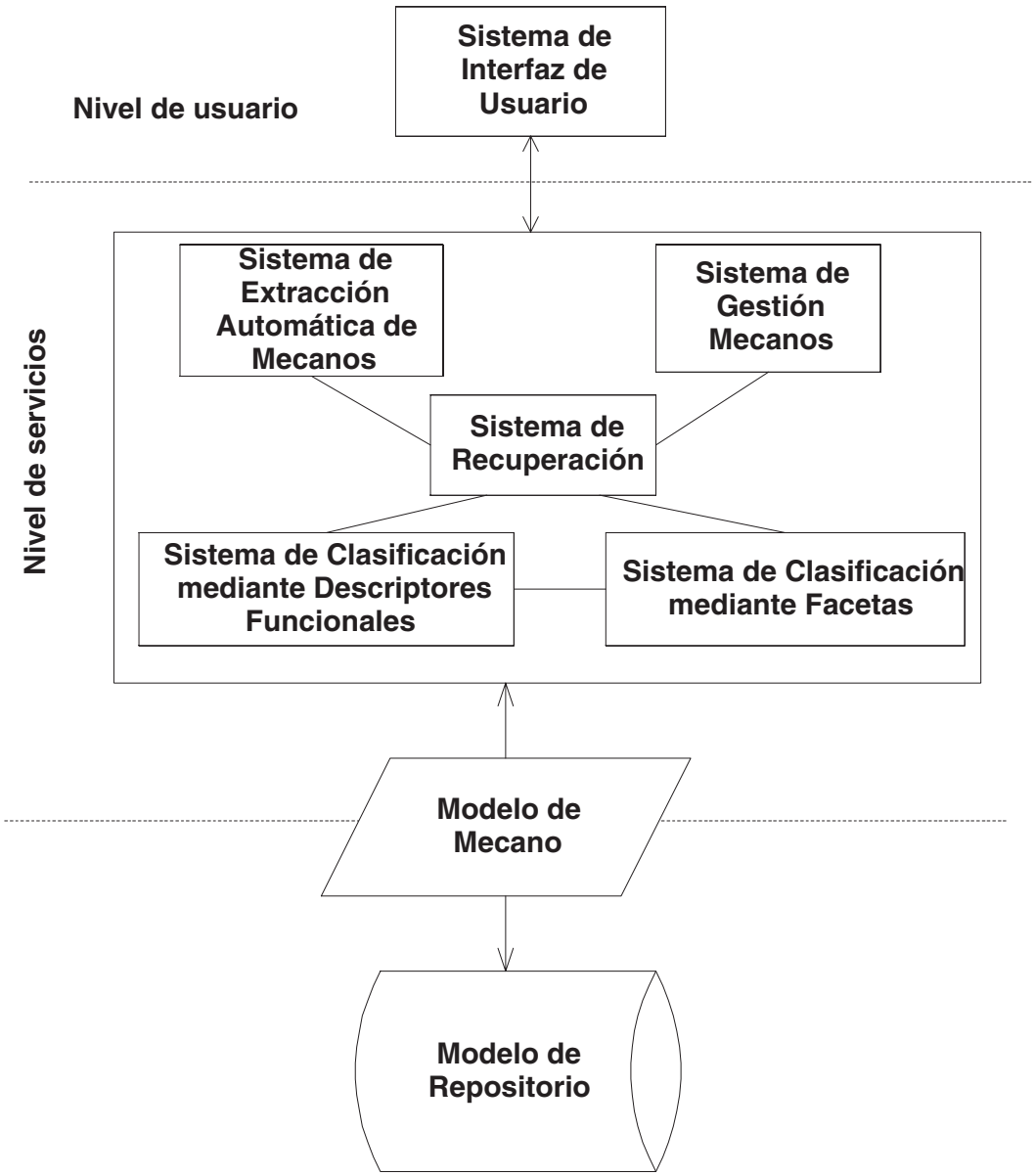

Figura 5.2: Arquitectura del entorno operativo para la reutilización.

Los mecanos han sido definidos desde un punto de vista técnico de una forma abstracta, pudiendo dar lugar a las más diversas interpretaciones en función de cuáles sean y cómo se organicen sus componentes.

De las diferentes posibilidades donde los mecanos pueden aplicarse, se destacan tres por su importancia en el desarrollo de software actual:

- *Mecanos como frameworks:* Los mecanos pueden representar un *framework* orientado a objetos definido desde su especificación a su implementación, pasando por su diseño basado en patrones. Esta es una aproximación que se centra completamente en el desarrollo orientado a objetos más puro, y que se aleja de enfoques metodológicos concretos.

- *Mecanos como soporte al desarrollo basado en componentes:* El desarrollo de software basado en componentes tiene hoy en día suficientes exponentes (*ActiveX, Javabeans, CORBA, DCOM*) para justificar su éxito y su interés como línea de investigación. La idea de componente es la de un elemento software reutilizable clasificado en el nivel de abstracción de la implementación (*normalmente en formato binario*), y que se pretende reutilizar como una caja negra. Un mecano puede representar un componente cuando éste no se entiende únicamente como una simple caja negra en un formato binario, sino como un conjunto de assets que engloban desde su especificación a su formato binario (*si existe esta representación*) pasando por su diseño y su código fuente (*siguiendo la filosofía descrita por la aproximación de Catalysis [D'Souza and Wills, 1999]*).

  Según esto, un componente puede verse como un mecano que se compone de todos los assets que se han generado en la construcción del mismo. El proceso de reutilización basado en mecanos que representan componentes, conduce a la selección del mecano o mecanos de la biblioteca y a su recuperación, de forma que el desarrollador con reutilización puede utilizarlo como una caja negra, o bien adaptarlo pues cuenta con toda la información disponible sobre el desarrollo del componente.

- *Mecanos formando parte de una línea de producto:* Los mecanos, al ser elementos software reutilizables de grano grueso que soportan simultáneamente varios niveles de abstracción, se adaptan especialmente bien a un enfoque de reutilización vertical centrada en dominios de aplicación, y más particularmente en líneas de producto. Así, un mecano vendría a representar tanto la definición de la línea de producto, como cada uno de los productos reutilizables que la nutren.

Aunque estas aproximaciones pueden convivir dentro de una organización que desee llevar a la práctica el modelo de reutilización propuesto, la que a priori más ventajas ofrece en pro de la reutilización sistemática es la basada en líneas de productos[2], por lo que a continuación, y desde un punto vista que relaciona la parte técnica con la parte de proceso, se va a presentar la relación entre mecanos y líneas de productos.

[2]Como se argumenta en el siguiente apartado al hablar del modelo de proceso.

5.2.1 Relación entre los Mecanos y las Líneas de Producto

Si se ve en un mecano una aproximación multinivel de abstracción a un dominio de aplicación, se entrará en la ingeniería de dominios. Sin embargo, hacer que un mecano modele un dominio de aplicación completo daría lugar a un elemento reutilizable demasiado grueso que perdería capacidad de reutilización. Por este motivo, una de las principales tareas dentro del modelo de proceso de reutilización es el estudio del dominio (*o dominios*) en el que una organización desea actuar, identificando sus subdominios (*líneas de producto*), sobre los cuales se quiere actuar.

Una línea de producto puede representarse por una serie de mecanos. Inicialmente, una línea de producto básica estará formada por su especificación y por su arquitectura [Cohen et al., 1996] que desde un punto de vista de granularidad más fina no va a ser otra cosa que el conjunto de assets de la línea de producto base con sus interrelaciones. Con este mecano inicial se representa la línea de producto, siendo un elemento reutilizable más en el repositorio. A partir de esta línea base se pueden empezar a desarrollar los productos (*aplicaciones*) que alimentan la línea de producto; bien utilizando la arquitectura común definida en la línea de producto y adaptándola con los requisitos específicos o bien creando nuevos sistemas mediante la composición y posterior adaptación de los assets de la línea de producto.

Con los nuevos productos creados se puede ir poblando la parte del repositorio donde se almacena la línea de producto, de forma que estos nuevos productos serán mecanos que comparten componentes de los que inicialmente sirvieron para crear la línea de producto base, y que todos ellos en conjunto forman la familia de productos que definen la línea de producto.

La figura 5.3 presenta las relaciones existentes entre los conceptos de dominio, línea de producto y mecano. Buscando la simplicidad del modelo se han considerado algunas restricciones que no tienen porque darse en situaciones reales, estas restricciones vienen dadas por las cardinalidades entre algunas entidades, de forma que se ha establecido que una línea de producto sólo puede pertenecer a un dominio, un producto sólo puede originarse de una línea de producto básica y por consiguiente formar parte de una sola línea de producto.

Bajo esta aproximación un mecano se identifica como un producto de una línea de producto. Esto incide en la creación de un proceso de introducción y recuperación de mecanos basado por el dominio y la línea de producto a la que pertenece.

Además, el trabajo con mecanos de una misma línea de productos permite que el proceso de reutilización, desde el punto de vista del desarrollador con reutilización, se

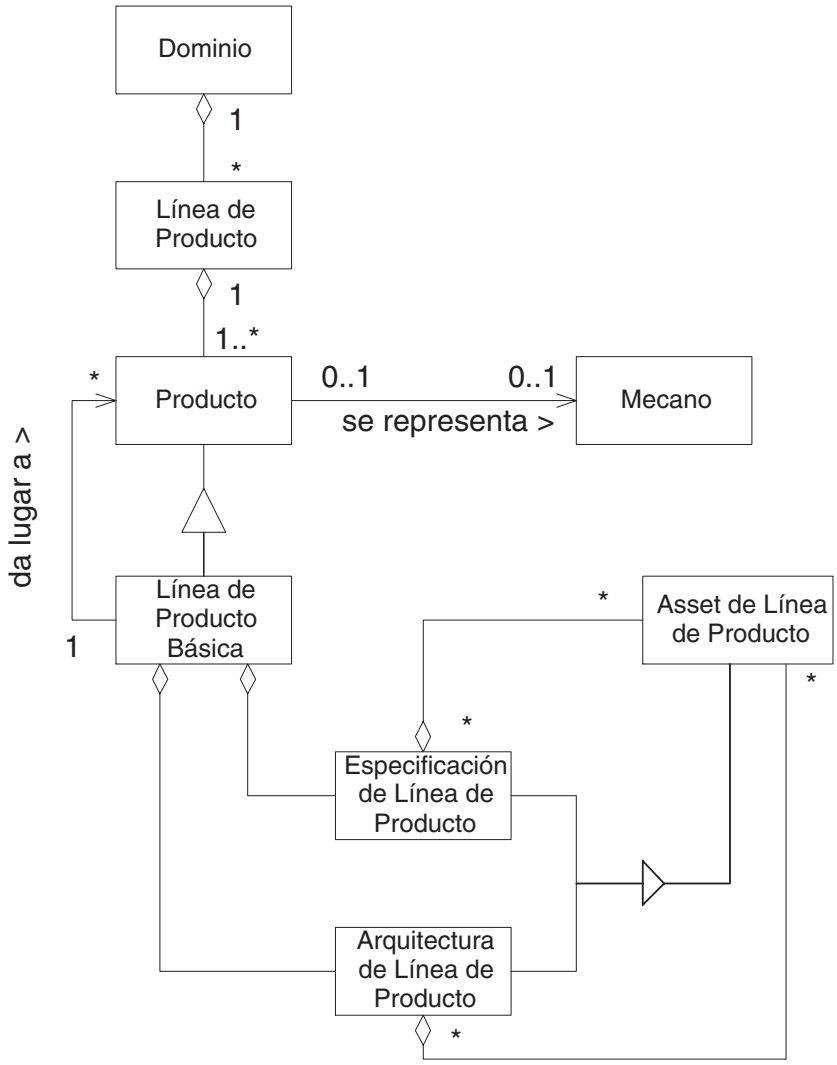

Figura 5.3: Dominios, líneas de producto y mecanos.

enriquezca con un enfoque de composición automática de mecanos, de forma que si un desarrollador con reutilización no encuentra ningún mecano que satisfaga sus necesidades de reutilización, se pueda obtener uno nuevo que se ajuste lo más posible a lo que necesita.

El acercamiento de los mecanos a las líneas de producto tiene otras consecuencias. La primera de ellas se encuentra en la vertiente técnica del modelo, porque ahora el proceso de diseño de mecanos en el desarrollo para reutilización vendrá guiado por el diseño de una línea de producto, para lo cual se tiene una excelente guía en [Clements et al., 1998].

Sin embargo, las líneas de producto unen a su parte técnica una parte de organización

empresarial (*negocios, organización, personal, gestión de recursos, aspectos culturales...*) que se pueden considerar al menos tan críticos como los técnicos [Brownsword and Clements, 1996].

5.3 Modelo de Proceso de Reutilización

El modelo de proceso de reutilización es el encargado de la organización de todas las actividades, recursos y procedimientos en relación con la política de reutilización de un organismo.

Una de las partes más relevantes del modelo de proceso es la que se ocupa de definir los procesos de creación de mecanos, que vienen diferenciados por un enfoque completamente composicional en el desarrollo para reutilización, donde los mecanos *persistentes* son compuestos *manualmente* por un desarrollador para reutilización; y por un enfoque híbrido composición/generación automático cuando, en el desarrollo con reutilización, los mecanos *persistentes* existentes no satisfacen las necesidades del desarrollador con reutilización.

En el caso en que sólo se utilicen mecanos *persistentes*, el proceso de reutilización seguido (*figura 5.4*) se asemeja mucho al proceso general, donde sólo cambia el hecho de que los elementos que se producen y se reutilizan son los elementos reutilizables de grano grueso denominados mecanos, con sus características intrínsecas que influyen en las actividades propias de este proceso.

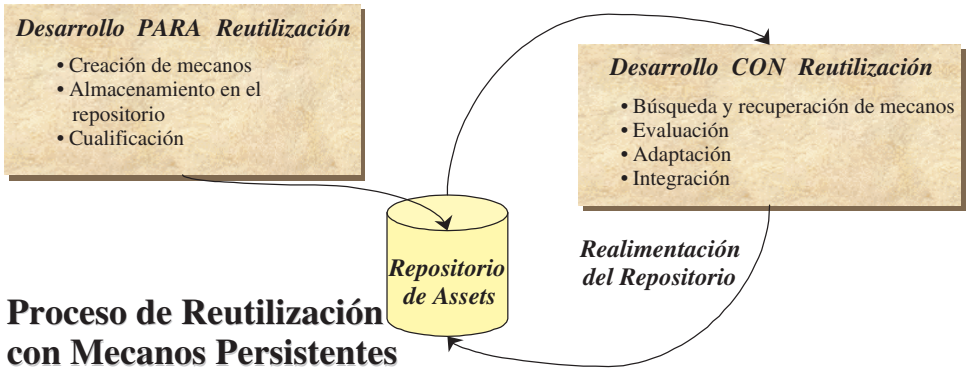

Figura 5.4: Proceso de reutilización con mecanos persistentes.

La cualificación de los mecanos es un aspecto fundamental dentro del MRG porque supone el sello de calidad con el que el desarrollador con reutilización espera contar a la hora de seleccionarlo para aplicarlo en un nuevo desarrollo. El marco general del proceso

de cualificación se trata en el modelo de cualificación de reutilización del MRG.

Para la búsqueda y recuperación de mecanos persistentes se depende de los servicios que ofrezca la biblioteca de reutilización, aunque deben estar basados en la combinación de métodos de búsqueda, que utilicen la información que ofrecen los mecanos y sus componentes, con métodos de navegación interactiva a través de la malla de enlaces que forman los componentes del mecano.

El proceso de reutilización se puede cerrar con una realimentación de nuevos mecanos, que tras la oportuna adaptación de los recuperados, se conviertan en nuevos elementos reutilizables.

Sin embargo, este proceso de reutilización se va a ver alterado cuando el desarrollador con reutilización decida generar por composición automática un nuevo mecano, como se puede apreciar en la figura 5.5.

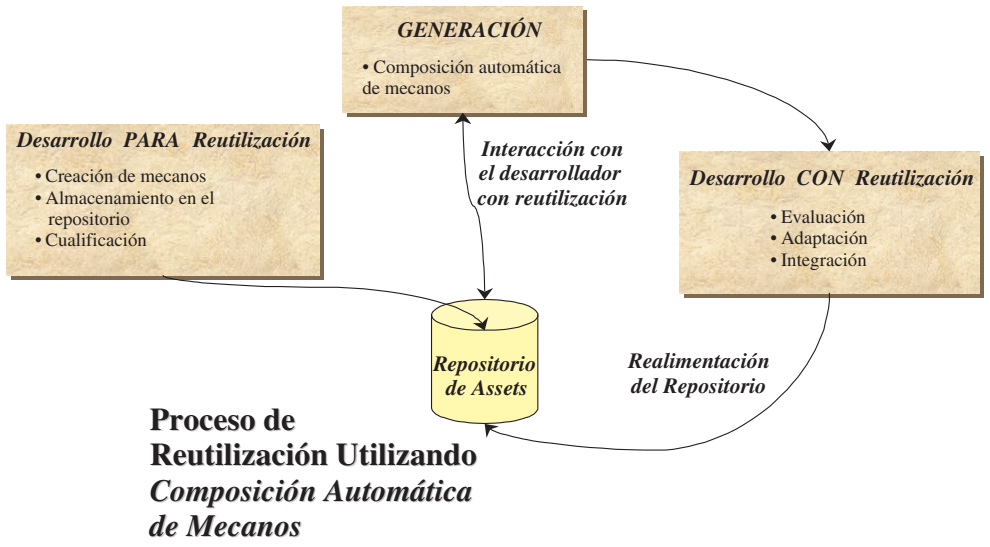

Figura 5.5: Proceso de reutilización con mecanos generados por composición automática.

En este caso, la parte de desarrollo para reutilización no varía, estando todos las modificaciones en el apartado de desarrollo con reutilización. La actividad de búsqueda y recuperación de mecanos desaparece de la fase de desarrollo con reutilización; apareciendo una nueva fase que se ha denominado *Generación*, en la que tras una gran interacción con el desarrollador con reutilización, se va a llevar a cabo el procedimiento de composición automática del mecano, tal como fue explicado en el *Capítulo 4*.

Esta nueva configuración del proceso de reutilización implica una mayor carga de traba-

jo en las actividades de adaptación e integración, porque no se puede obviar que el mecano generado es una aproximación a lo que el desarrollador con reutilización está buscando, además de tener que discernir y evaluar las diferentes alternativas que incluye.

En general, aunque un proceso de reutilización sistemática del software dentro de un organismo puedan llevarse a cabo desde muy diversas perspectivas, existe siempre un patrón que se repite (*figura 5.6*): un productor (*o desarrollador para reutilización*) desarrolla productos software reutilizables, los hace públicos y un consumidor (*o desarrollador con reutilización*) los recupera y utiliza para construir sistemas software finales.

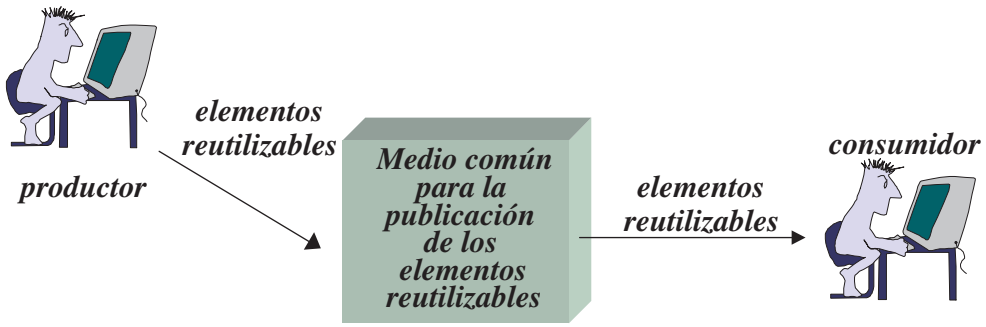

Figura 5.6: Patrón general del proceso de reutilización del software.

En cada caso concreto el patrón de la figura 5.6 se adapta a las necesidades particulares de cada organización. Así, el productor puede ser ajeno a la organización y desarrollar COTS (*Commercial Off-The-Shelf*) que la organización adquiere y explota en sus aplicaciones, o por el contrario el productor está integrado en el organigrama de la organización, desarrollando elementos reutilizables para el uso interno de la misma, e incluso el rol de productor y consumidor pueden llegar a coincidir cuando el desarrollador con reutilización adapta los elementos reutilizables y realimenta el espacio común de publicación de elementos reutilizables.

En el caso concreto del MRG tanto el proceso de reutilización con mecanos persistentes (*figura 5.4*) como el proceso de reutilización con mecanos generados por composición automática (*figura 5.5*) se ajustan a este patrón genérico.

El modelo de proceso también debe ocuparse de cuidar las actividades de desarrollo con reutilización, ya que tras la extracción de los elementos de reutilizables del repositorio comienza el trabajo de entendimiento, adaptación e integración de los mismos en los nuevos desarrollos. En este sentido se distinguen dos casos diferentes.

En el primero el desarrollo para reutilización y el desarrollo con reutilización recaen

dentro de la misma organización, con lo que se deben establecer las actividades oportunas de realimentación del repositorio y la recogida de las métricas oportunas para poder cualificar el proceso de reutilización (*lo que se define en el modelo de cualificación de la reutilización*).

El segundo caso se da cuando el desarrollo para y con reutilización recaen en diferentes organizaciones, de manera que la organización que desarrolla con reutilización tiene que establecer los criterios de adquisición del software oportunos para obtener los elementos reutilizables de terceras partes. En este sentido ya se están proponiendo extensiones al modelo **CMM** (*Capability Maturity Model*) del **SEI** (*Software Engineering Institute*) para soportar la incorporación de líneas de producto y adquisición de COTS entre otras facetas de la reutilización del software [Ragan and Reifer, 1998].

Sin embargo, para que de este enfoque de reutilización sistemática se obtengan los mayores beneficios deben cuidarse los detalles de organización de todos los recursos involucrados en el proceso de reutilización. Esta organización incluye a los elementos reutilizables, para lo cual es indispensable conocer qué significan los mecanos para el organismo que los usa.

De las diversas aproximaciones con las que se puede entender los elementos reutilizables, y que se discutieron en el modelo técnico, desde aquí se opta por potenciar la aproximación relacionada con las líneas de producto. Esta decisión se justifica en el hecho de que las líneas de producto se convierten en uno de los caminos más adecuados para introducir la reutilización sistemática en una organización, además de ser una de las formas de reutilización que más beneficios aporta [Clements, 1997b], [Maymir-Ducharme, 1997], [Griss et al., 1998]. Pero, también se apoya en la naturalidad con la que un mecano se ajusta al enfoque arquitectónico de las líneas de producto, y en las ventajas que, de una organización de una biblioteca de reutilización en líneas de producto, se derivan para la reducción del espacio inicial de trabajo en el enfoque de generación de mecanos por composición automática de assets.

5.4 Modelo de Cualificación de Reutilización

La calidad es uno de los objetivos que persiguen los desarrolladores de software, pues han asumido que es el medio que conduce a una mejora de la productividad y de los costes, satisfacción del cliente y cumplimiento de plazos.

En reutilización la calidad de los elementos reutilizables juega, si cabe, un papel más decisivo que en los desarrollos sin reutilización. En el desarrollo para reutilización, la creación de elementos reutilizable tiene como objetivo el desarrollo de elementos de alta

calidad que sean reutilizados el mayor número de veces para amortizar así el incremento en los costes derivado del proceso de reutilización. Sin embargo, en el desarrollo con reutilización la calidad se convierte no sólo en uno de los objetivos del producto final, sino en una premisa que deben cumplir los bloques reutilizables que se utilicen para la construcción de dicho software.

En otras palabras, el desarrollador con reutilización necesita la certificación de que lo que va a reutilizar cumple unos mínimos de calidad. Esta es la misión del modelo de cualificación de reutilización del MRG, para lo cual va a establecer un *plan de calidad* y un *plan de métricas*. Además, se establece un *esquema de auditoría* como filtro previo a cualquier otra actividad de cualificación de los componentes de los mecanos.

El plan de calidad se centra en la reusabilidad y en la fiabilidad, factores que se han considerado definitorios de la calidad de los assets [Manso et al., 1998].

El plan de métricas se deriva del propio plan de calidad y va a establecer el conjunto de métricas que van a permitir calcular medidas de los productos obtenidos (*elementos reutilizables: mecanos y assets*) y del proceso de reutilización. Así, las métricas que se contemplan deben cuantificar como mínimo la reusabilidad de un elemento reutilizable, la reutilización llevada a cabo de un elemento reutilizable, el aumento de calidad de un producto cuando se desarrolla con reutilización y los cambios/mejoras en el proceso de desarrollo con/para reutilización [Manso et al., 1999a].

El proceso de auditoría va a permitir una evaluación independiente de los productos y procesos software, para certificar el grado de cumplimiento de los objetivos previamente establecidos. Teniendo en cuenta la tipología de los assets que integran un mecano, el objetivo fundamental de las auditorías es el de controlar y certificar los documentos que representan a estos assets, con anterioridad a su almacenamiento en el repositorio [Manso et al., 1999b].

El modelo de cualificación de reutilización propuesto establece un marco inicial que juega el rol de intermediario entre el producto (*elemento reutilizable*) y el cliente (*repositorio*), del que se tienen unas primeras aproximaciones en [Manso et al., 1998] y en [Manso et al., 1999a], permitiendo establecer:

- Qué necesidades deben ser satisfechas desde el punto de vista del repositorio, o aspectos de la calidad que serán relevantes cuando el elemento vaya a reutilizarse. Es evidente que un elemento reutilizable que no es fiable o no está bien documentado no satisfará a los usuarios del repositorio y por tanto no será apto para ser reutilizado [Frakes and Terry, 1996].

- Cómo medir hasta que punto las necesidades han sido satisfechas, o en qué grado serán cubiertos los aspectos de la calidad relevantes para la reutilización.

5.4.1 Proceso de Cualificación de los Assets

La cualificación de un mecano, como elemento de grano grueso que es, debe pasar por que todos los assets que lo integran hayan sido cualificados.

El proceso de cualificación de los assets se lleva a cabo al margen del desarrollo de los mismos, justo en el momento en que el producto software va a ser incorporado al repositorio. Su finalidad es determinar en qué medida dicho producto cumple con los objetivos de calidad fijados en el repositorio.

Estas peculiaridades van a determinar el tipo de proceso a utilizar y las actividades asociadas.

Este proceso debe estar soportado por herramientas que faciliten la cualificación automática de los assets. Algunas de ellas formarán parte de los servicios de la biblioteca de reutilización, mientras que otras pueden ser externas a ésta (*incluso lejanas geográficamente de la biblioteca de reutilización*). Las diferentes fases del proceso se representan en la figura 5.7.

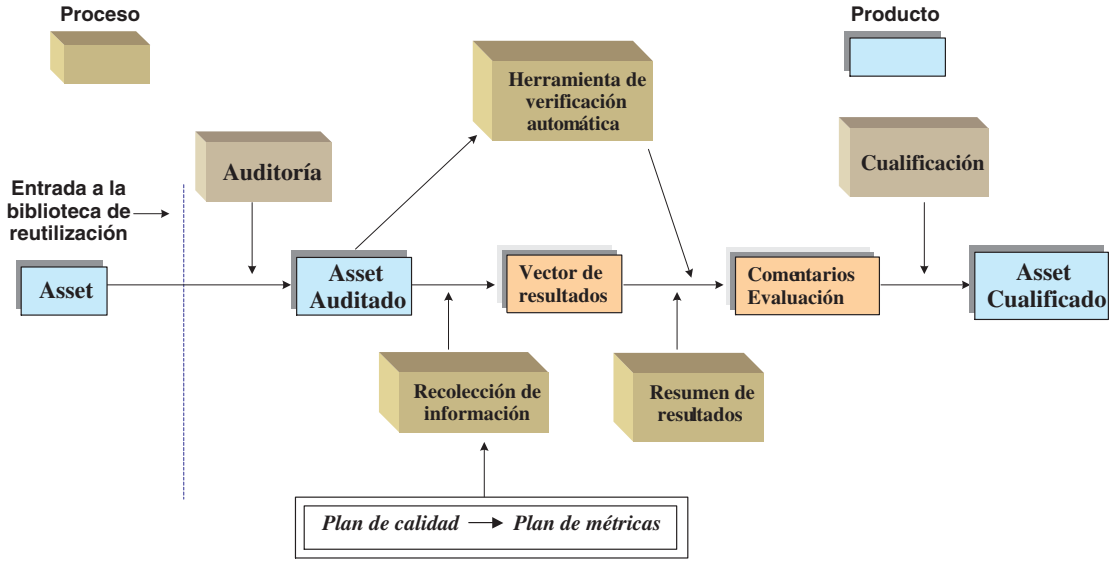

Figura 5.7: Proceso de cualificación de assets.

Los objetivos y actividades a desarrollar en cada fase para alcanzarlos son:

- **Fase 1 de Auditoría:** El objetivo principal de esta fase es controlar el tipo y calidad de la documentación del asset. La entrada a esta fase es el nuevo *asset* y la salida el *asset-auditado* según la norma **IEEE std. 1028-1988** [IEEE, 1994]. En él se incluye el registro que recoge el resultado de la auditoría, que formará parte de la información que se debe utilizar para la cualificación.

- **Fase 2 de Recogida de Resultados:** La entrada es el *asset-auditado* y la salida un *vector de resultados* que recoge las medidas resultantes de aplicarle el plan de métricas. En esta fase se realizan las mediciones del asset-auditado que vengan establecidas por el plan de métricas; el cual se elige en función de la fase del ciclo de vida a la que pertenezca el asset y del paradigma con el que se haya desarrollado.

  El plan de métricas se elabora a partir de los factores de calidad considerados: *fiabilidad* y *reusabilidad*, siguiendo fundalmentalmente el modelo **FCM** (*Factor, Criterion, Metric*). Este modelo permite descomponer los factores que no son directamente mensurables (*cohesión, acoplamiento...*), en otros atributos del software que sí lo son. Así, cada atributo o factor de calidad tendrá asociado un vector de métricas, supuestamente relacionados con él. A partir de estos vectores se obtienen los vectores de mediciones concretas para el asset en estudio.

- **Fase 3 de Valoración de Resultados:** La entrada es el *vector de resultados* y la salida es una *valoración de los diferentes aspectos de la calidad y comentarios sobre las posibles causas de una valoración inaceptable.* Una vez obtenido el vector de resultados, V, que corresponde a un atributo del elemento reutilizable, el propio proceso produce una valoración automática de dicho vector de acuerdo con las siguientes reglas:

  - ◇ Sea V_i la componente i-ésima del vector asociado al atributo A.
  - ◇ C_i es el rango crítico de valores que permite clasificar al elemento de software en aceptable, marginal o inaceptable con respecto a la componente V_i.
  - ◇ I_i es un indicador de aceptación que toma valor 1 en el caso de que el software sea aceptable con respecto a la métrica i del atributo A y 0 en otro caso. O lo que es igual:
    - $V_i \in C_i \Leftrightarrow$ el asset es aceptable con respecto a $V_i \Leftrightarrow I_i = 1$

  Para valorar la calidad del atributo A se utiliza un modelo multiplicativo de la siguiente forma: $Q(A) = \prod I_i$. Si $Q(A) = 1$ el asset es aceptable con respecto al

atributo A, de lo contrario, se genera un conjunto de comentarios explicando las posibles causas de este hecho.

Esta valoración por definición es demasiado exigente (*cualifica como aceptables un subconjunto muy restringido del conjunto de elementos aceptables*) y no utiliza toda la información que las métricas proporcionan. Como complemento se plantea la utilización de un modelo lineal más informativo: $S(A) = \sum \alpha_i \,.\, V_i$.

- **Fase 4 de Cualificación:** Las entradas son el *registro de la auditoría, el vector de resultados* y *las valoraciones de la Fase 3.* A partir de esta información se cualifica el asset siguiendo un primer modelo lineal, $S(F) = \sum \beta_j \,.\, S(A_j)$, que permite obtener la métrica del factor F como suma ponderada de las métricas de atributos asociados a él. La validación de dicho modelo y de su capacidad predictiva se hará posteriormente, lo que dará lugar, posiblemente, a un reajuste del vector de métricas y a cambios en el modelo. La cualificación del asset, en principio, comprenderá toda esta información, esto es, la de entrada a esta fase y la generada por el modelo que sintetiza parte de la información.

5.4.2 Auditoría

El objetivo que se ha tenido en cuenta al desarrollar las auditorías ha sido controlar la documentación que llega a la biblioteca de reutilización, siguiendo las pautas marcadas por el plan de calidad del mismo.

Se ha entendido por auditoría, siguiendo la normativa de la IEEE [IEEE, 1994], *la evaluación independiente de productos o procesos de software que permite determinar el cumplimiento con estándares, líneas directrices, especificaciones y procedimientos siguiendo criterios objetivos.* En éstos se deben especificar:

- Forma y contenido de los productos a desarrollar.
- Proceso para hacerlo.
- Cómo se medirá el cumplimiento con estándares y líneas directrices.

Las auditorías que se han realizado hasta el momento han tenido como objetivo certificar que la documentación era correcta y completa, en el siguiente sentido:

- Completa, es decir, están presentes todos los documentos que el método utilizado para desarrollar el producto exige. Sin olvidar además documentos relativos a actividades de verificación y validación.
- Correcta, en cuanto a la sintaxis o representación del documento, siguiendo el método utilizado como patrón.

Los elementos de una auditoría, siguiendo la norma IEEE [IEEE, 1994], son:

- Entradas, o elementos de deben estar disponibles al comienzo de la auditoría.
- Salidas, o documentos que la auditoría debe generar.
- Criterios de salida o condiciones exigidas para dar por finalizada la auditoría.

Como elementos de entrada, además del propósito y ámbito de la auditoría, se debe contar con elementos y procesos a auditar y su historial, y se dispondrá de las listas de chequeo que son el núcleo de la auditoría. El tipo de listas de chequeo que se van a utilizar tienen que ver con los tipos de assets que hay que auditar.

El esquema que se ha seguido para construir las listas de chequeo ha sido:

- Identificación del modelo a que se refieren.
- Lista de elementos estándares del modelo, desde el punto de vista sintáctico.
- Asociación de métricas para medir los diferentes atributos de cada elemento de la lista.

El rango de valores de las métricas, 0, 1, 2 tratan de diferenciar entre los elementos que no han seguido el estándar, pero son inteligibles (*1*) porque siguen un estándar explícito o muy evidente, y los que no siguen ningún estándar (*2*). El cero se asocia a los elementos que siguen el estándar de la lista de chequeo.

Los criterios de salida que deben considerarse para determinar si la auditoría ha acabado o no son los siguientes: el asset se ha certificado y los informes sobre el asset y la propia auditoría están hechos. Además, si es posible localizar al autor del asset, éste deberá haber hecho las correcciones pertinentes para que el asset sea auditado de nuevo, hasta corregir los errores.

La información de salida de la auditoría va a ser fundamentalmente de dos tipos:

- *Información sobre el asset:* Contiene un resumen con las conclusiones y clasificación de errores, que se añadirá a la documentación del mismo, lo que produce un incremento en la calidad del asset. En la información relativa al asset se debe incluir la certificación del asset en: ***aceptado, aceptado con condiciones*** o ***no aceptado***, de acuerdo con las normas siguientes:

  - ◇ Cada métrica X_{ik} (*donde* X_{ik} *hace referencia al atributo que se mide,* i*, y al asset en el que se está midiendo,* k) interviene en el cálculo de la distancia entre el asset y un asset ideal, que cumple totalmente el estándar de que se trata, a través de su distribución de frecuencias $[F(X_{ik}(j)),\ j = 0..2]$.
  - ◇ Teniendo en cuenta que la frecuencia absoluta del valor j en el asset k es:

$$F_k(j) = \sum_i F(X_{ik}(j)) \tag{5.1}$$

    se define la distancia de la siguiente forma:

$$d(asset_k) = 1 - \left(\frac{\sum_i F(X_{ik}(0))}{\sum_j F_k(j)} \right) \tag{5.2}$$

  - ◇ El algoritmo para certificar el asset se ha diseñado teniendo en cuenta que si alguna métrica X_{ik} toma el valor 2, el asset tiene aspectos que son ininteligibles:

    Si $d(asset_k) = 0$ **entonces** la cualificación es *aceptado.*

    Sino si $F_k(2) = 0$ **entonces** la cualificación es *aceptado con condiciones.*

    Sino la cualificación es *no aceptado.*

- *Información sobre la propia auditoría:* Que contiene un resumen del esfuerzo de la auditoría, una clasificación de errores y conclusiones de la auditoría, con recomendaciones tanto para los auditores (*mejora de listas...*) como para los suministradores.

En general, contar con un proceso de auditoría dentro del modelo de cualificación supone un sello de madurez, que garantiza al usuario del repositorio, *desarrollador con reutilización*, productos comprensibles y adaptables. Además, el tipo de auditoría antes mencionado es imprescindible se se quieren utilizar herramientas automáticas, basadas en estándares de documentación para los productos software.

5.4.3 Cualificación de Mecanos

Los mecanos son elementos reutilizables de grano grueso compuestos por assets clasificados en diferentes niveles de abstracción. Esto hace que la cualificación de los mismos presente las siguientes características:

1. La cualificación de un mecano se dará por niveles de abstracción, en función de la cualificación de los assets que formen parte de cada nivel de abstracción.

2. Teniendo en cuenta los objetivos de calidad que guían la cualificación de los assets, se deducen los que guiarán la cualificación de los mecanos.

3. La reusabilidad real de los mecanos será una de la medidas de mayor interés pues servirá para evaluar la utilidad de los mismos y, en último término, la del propio repositorio.

La definición completa de la cualificación de los mecanos es la parte central del Modelo de Cualificación de Reutilización. Todavía queda mucho trabajo por hacer en esta parcela, constituyéndose así en una de las líneas de trabajo futuro más importantes que deja abiertas esta tesis.

Capítulo 6

Conclusiones y Trabajo Futuro

La aportación teórica de este trabajo ha consistido en el desarrollo de una estructura compleja de reutilización, que recibe el nombre de mecano, de un proceso de construcción de estos mecanos y de un marco general para un modelo de reutilización sistemática del software basado en mecanos.

La definición de la estructura de reutilización se ha realizado con vistas a potenciar el proceso de reutilización sistemática, para lo que se ha optado por trabajar con elementos reutilizables de grano grueso, formados por una malla de assets que ofrecen una cobertura a todo el ciclo de desarrollo, al estar clasificados en diferentes niveles de abstracción.

El proceso de construcción de mecanos se ha desarrollado para que automáticamente se asegure la consecución de un mecano bien formado. Los mecanos se pueden construir de acuerdo a dos posibles situaciones: *construcción manual* en el desarrollo para reutilización, de forma que el proceso guíe al desarrollador para reutilización en su cometido, impidiendo el establecimiento de estructuras no permitidas; *construcción automática* en el desarrollo con reutilización, de manera que el proceso, a partir de los datos introducidos por el desarrollador con reutilización, extraiga los assets necesarios, junto a sus relaciones, para componer automáticamente el mecano bien formado que mejor se ajuste a los requisitos del reutilizador.

La presencia de los mecanismos de construcción manual y automática confieren al proceso de construcción general de mecanos un matiz híbrido o mixto de las tecnologías de composición y generación.

El modelo de reutilización basado en mecanos recoge las líneas directrices para que cualquier organización ponga en marcha un plan de reutilización sistemática del software.

En él se han identificado tres ejes fundamentales: el *eje técnico*, que básicamente recoge la definición y manipulación de los mecanos, así como todo lo referente al soporte operativo de la reutilización; el *eje de proceso*, que se encarga de establecer las actividades a realizar en el desarrollo *para* y *con* reutilización, además de dotar a una organización de los recursos humanos y técnicos en torno al proceso general de reutilización; y por último el *eje de cualificación*, que se centra en los aspectos de auditoría y certificación de los elementos reutilizables, para que bajo la supervisión de un plan de calidad global, el desarrollador con reutilización tenga un referente de la calidad de los elementos con los que va construir nuevos sistemas software.

Para la consecución de estos objetivos se han desarrollado los siguientes resultados concretos:

1. Se ha definido el Modelo de Mecano, que aúna el modelo de componente reutilizable, mecano, y el modelo de repositorio que permite almacenar mecanos junto con toda la información necesaria para su tratamiento y recuperación. Este Modelo de Mecano se ha definido utilizando técnicas semi-formales basadas en UML 1.x.

2. En la definición del modelo de componente reutilizable (*assets clasificados en diferentes niveles de abstracción y relacionados entre sí para formar el elemento reutilizable de grano grueso denominado mecano*) ha sido fundamental describir:

   - Los niveles de abstracción que se iban a contemplar (**requisitos, diseño e implementación**).
   - La estructura del asset componente.
   - Las posibles relaciones semánticas que podían establecerse entre los assets componentes, cuyas características vienen dictadas por un conjunto reducido de relaciones estructurales (**reificación, agregación, composición, extensión, usa a, asociación fuerte y asociación débil**).

3. Se ha definido formalmente el modelo de componente reutilizable soportado por la teoría de grafos. En concreto, se ha utilizado una variante de los grafos coloreados, *los grafos tubo*, para este cometido. Este modelo formal, define lo que se considera un mecano bien formado, que está basado en la propiedad de que todas las aristas del grafo subyacente están bien formadas, es decir, cumplen alguna de las reglas de formación que se han establecido en este modelo formal.

4. Se ha construido una gramática de grafos dependiente del contexto, G_M, cuyas reglas de producción permiten construir mecanos bien formados, para lo que se ha demos-

trado que el lenguaje generado por la gramática es equivalente al conjunto de todos los mecanos bien formados.

5. Se ha enunciado un proceso que, dirigido por la gramática de grafos G_M, guíe la construcción/verificación de los mecanos construidos "*manualmente*" dentro del proceso de desarrollo para reutilización.

6. Se ha enunciado un proceso que, sobre un contexto fijado por el desarrollador con reutilización y dirigido por la gramática de grafos G_M, es capaz de extraer automáticamente assets componentes de otros mecanos para generar por composición un nuevo mecano que se ajuste, en la medida de lo posible, a los requisitos de reutilizador, dentro del proceso de desarrollo con reutilización.

7. Se ha establecido un marco general para un modelo de reutilización basado en mecanos, en el que se han identificado tres líneas ortogonales de actuación:

   - La vertiente técnica donde se ha desarrollado la definición y manipulación de los mecanos, se ha establecido el soporte operativo para la reutilización y se ha definido la relación existente entre los mecanos y las líneas de producto (*modelo técnico de reutilización).*
   - La vertiente de proceso donde se ha adaptado el proceso general de reutilización, formado por la parte de desarrollo para reutilización y la parte de desarrollo con reutilización, al caso particular de los mecanos. Además, se ha ocupado de identificar los aspectos organizativos de la reutilización, para lo que se ha propuesto una política basada en líneas de productos, que aprovecha la relación sinérgica definida en el modelo técnico (*modelo de proceso de reutilización*).
   - La vertiente de cualificación encargada de velar por la calidad de los elementos reutilizables, donde se ha establecido un proceso de auditoría y un plan de calidad, con las métricas adecuadas, para certificar la calidad de los assets componentes de los mecanos (*modelo de cualificación de reutilización*).

6.1 Líneas de Investigación Futuras

Alcanzados los objetivos propuestos en la realización de este trabajo de investigación, se considera necesario proponer una serie de líneas de investigación, como continuación y profundización de la labor realizada. En este sentido, tres son los aspectos que interesa abordar. El primero está relacionado con el proceso de composición automática de mecanos

en tiempo de reutilización. El segundo hace referencia al modelo de cualificación de reutilización. Y el tercero es el relacionado con el desarrollo e implantación de los resultados obtenidos.

El mecanismo de extracción/composición automática de mecanos ofrece siempre como resultado un mecano bien formado, al estar basado en la gramática de grafos dependiente de contexto G_M. Pero es susceptible de ser mejorado, especialmente con la incorporación de un tratamiento más adecuado de las alternativas que se producen. Otra mejora es contemplar aspectos de relaciones entre las descripciones de los requisitos funcionales que desencadenan el proceso, especialmente en lo tocante a temas de incompatibilidades entre requisitos; para lo que se pueden buscar soluciones ortogonales al modelo establecido, embebidas en diferentes servicios del soporte operativo, para lo que **FODA** (*Feature-Oriented Domain Analysis*) [Kang et al., 1990] o **MRAM** (*Method for Requirements Authoring and Management*) [Mannion et al., 1999] son referencias que ofrecen interesantes caminos a estudiar.

En este trabajo se ha propuesto un marco general para el proceso de cualificación de reutilización, en el que se establece cuál es su cometido. Pero el trabajo realizado en la definición de sus procesos está en un estado muy incipiente, habiéndose dado los primeros pasos en la definición de la fase de auditoría y del plan de métricas para assets, quedando un largo camino por recorrer hasta cerrar todos estos aspectos para los mecanos.

En lo referente al desarrollo e implantación de herramientas que hagan uso de los conceptos teóricos desarrollados, en el *Apéndice 1* se hace un repaso de las experiencias llevadas a cabo en este trabajo, aunque, como se deduce de la lectura de dicho apéndice, existe un desfase entre los avances teóricos y los prototipos realizados, que debe ser solucionado en los siguientes prototipos, con un especial hincapié en las técnicas de composición automática de mecanos en tiempo de reutilización.

6.2 Publicaciones Relacionadas con este Trabajo

Durante el período de realización de esta propuesta de tesis doctoral los resultados parciales se han ido presentando en diversos foros, tanto nacionales como internacionales, lo que ha dado lugar a la siguiente serie de publicaciones.

Capítulos en libros

1. **Manso Martínez, Mł Esperanza y García Peñalvo, Francisco José**. "*Medición en la Reutilización Orientada a Objetos*". Capítulo 6 en José Javier Dolado Cosín y Luis Fernández Sanz (coordinadores). *Medición para la Gestión en la Ingeniería del Software*. Páginas 111-130. ISBN 84-7897-403-2. Ra-ma, 2000.

Comunicaciones a Congresos Nacionales

1. **García Peñalvo, Francisco José, Marqués Corral, José Manuel y Maudes Raedo, Jesús Manuel**. "*Mecano: Una Propuesta de Componente Software Reutilizable*". En las actas de las II Jornadas de Ingeniería del Software. Editores Oscar Díaz y Philippe Lopistéguy. (Donostia-San Sebastián, España, 3-5 septiembre de 1997): 232-244. 1997.

2. **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.

3. **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*". En las actas de las III Jornadas de Ingeniería del Software JIS'98. Editores Ambrosio Toval Álvarez y Joaquín Nicolás Ros. (Murcia, 11-13 de Noviembre de 1998):155-166. 1998.

4. **Romay Rodríguez, M. del Pilar, García Peñalvo, Francisco José, Crespo González-Carvajal, Yania y Laguna Serrano, Miguel Ángel**. "*Una experiencia Práctica de Reutilización: Puesta en Marcha del Repositorio GIRO*". En las actas de las III Jornadas de Trabajo MENHIR. Editores Begoña Moros y José Sáez. (Murcia 13-14 de Noviembre de 1998): 103-108. 1998.

5. **Manso Martínez, M. Esperanza, García Peñalvo, Francisco José, Rodríguez Díez, Juan José y Laguna Serrano, Miguel Ángel**. "*Modelo de Cualificación de Assets del Repositorio GIRO*". En las actas de las III Jornadas de Trabajo MENHIR. Editores Begoña Moros y José Sáez. (Murcia 13-14 de Noviembre de 1998): 109-114. 1998.

6. **Manso Martínez, Esperanza, Romay Rodríguez, M. Pilar and García Peñalvo, Francisco José**. "*Repository Asset Audit*". In proceedings of the 4th Workshop MENHIR. Francisco José García and José Manuel Marqués Editors. (Sedano - Burgos (Spain), May 6-7, 1999): 11-15. 1999.

7. **Sáez, José, Fernández, José Luis, Toval, Ambrosio, Manso, Esperanza, Laguna, Miguel Ángel and García, Francisco José**. "*A Reuse Model for Formally Verified UML Diagrams*". In proceedings of the 4th Workshop MENHIR. Francisco José García and José Manuel Marqués Editors. (Sedano - Burgos (Spain), May 6-7, 1999): 16-20. 1999.

8. **Manso Martínez, M. Esperanza, García Peñalvo, Francisco José, Romay Rodríguez, M. Pilar y Marqués Corral, José Manuel**. "*Modelo de Cualificación y Auditorías de Elementos Reutilizables de un Repositorio*". Actas de las IV Jornadas de Ingeniería del Software y Bases de Datos, JISDB'99. Editores Pere Botella, Juan Hernández y Félix Saltor. (Cáceres, 24-26 de Noviembre de 1999): 355-366. 1999.

9. **Prieto, Félix, Crespo, Yania, García, Francisco José y Laguna, Miguel Ángel**. "*Construcción de Frameworks Basada en Análisis de Conceptos Formales y Soportada por Mecanos*". En las actas de las V Jornadas de Trabajo MENHIR. Editoras Mł José Rodríguez Fórtiz y Patricia Paderewski Rodríguez. (Granada, 30-31 de Marzo de 2000): 36-47. Marzo, 2000.

Comunicaciones a Congresos Internacionales

1. **García Peñalvo, Francisco José, Marqués Corral, José Manuel y Maudes Raedo, Jesús Manuel**. "*Mecanos as Basis of a Compositional/Generative Mixed Reuse Model*". In proceedings of the second edition of the European Reuse Workshop (ERW'98 - Madrid, November, 1998). Vol(2): 17-20. 1998.

2. **García Peñalvo, Francisco José, Romay Rodríguez, M. del Pilar, Marqués Corral, José Manuel y Crespo González-Carvajal, Yania**. "*Mecanos: Exposición de Resultados y Líneas de Trabajo Abiertas en la Reutilización Sistemática del Software*". En las actas de las 2as Jornadas Iberoamericanas de la Ingeniería de Requisitos y Ambientes de Software IDEAS'99. Editores Oscar Pastor, Carlos González, Ignacio Trejos y Emilio Insfrán. (Alajuela-Costa Rica, 24-26 de Marzo 1999): 193-204. 1999.

3. **Genero, Marcela, Manso, M. Esperanza, Piattini, Mario and García, Francisco José**. "*Assessing the Quality and the Complexity of OMT Models*". In Proceedings of the 2nd European Software Measurement FESMA'99. M. Hooft van Huyduynen, G. Poels, B. Peeters and R. Nevalainen editors. (Amsterdam, The Netherlands, 4-8 October 1999): 99-109.

Technical Reports

1. **García, Francisco José, Marqués, José Manuel y Maudes, Jesús Manuel**. "*Análisis y Diseño Orientado al Objeto para Reutilización*". Technical Report (TR-GIRO-01-97V2.1.1), Universidad de Valladolid (España). Octubre 1997.

2. **García Peñalvo, Francisco José, Marqués Corral, José Manuel, Laguna, Miguel Ángel y Maudes Raedo, Jesús Manuel**. "*Mecanos Reutilizables Estáticos*". Technical Report (TR-GIRO-01-98V0.9), Universidad de Valladolid (España). Marzo, 1998.

3. **García Peñalvo, Francisco José, Marqués Corral, José Manuel, Laguna, Miguel Ángel y Maudes Raedo, Jesús Manuel**. "*Mecanos: Soporte de Diferentes Niveles*

de Abstracción en los Elementos Software Reutilizables". Technical Report (TR-GIRO-02-98V1.2.1), Universidad de Valladolid (España). Octubre, 1998.

Apéndice A

Experiencias Prácticas

El soporte operativo a la reutilización está soportado por una biblioteca de reutilización que entre otros importantes servicios ofrece el servicio de repositorio.

Uno de los objetivos que se perseguían al comenzar este trabajo de tesis doctoral era llevar las propuestas teóricas realizadas a entornos de desarrollo reales basados en la reutilización del software. De acuerdo con esto, de forma paralela a los avances en el terreno teórico o conceptual, se han ido realizando diversas experiencias para obtener entornos de reutilización experimentales donde implementar el concepto de mecano. El propio devenir de estas experiencias han registrado el proceso de evolución y maduración de este trabajo.

En este apéndice se va a hacer un recorrido por las experiencias prácticas más notables acaecidas en relación con la implementación y soporte de los mecanos en entornos de reutilización reales. Así, en la primera sección se describe una primera fase centrada en la adaptación de un motor de repositorios comercial como es el caso de **EUROWARE** [SER Consortium, 1996]; en la segunda sección se presenta el trabajo que se está realizando para la construcción de una biblioteca de reutilización que soporte de forma nativa el concepto de mecano; finalmente, la tercera sección cierra este apéndice con unas conclusiones y las líneas de trabajo futuro en este apartado práctico.

A.1 Experiencia con EUROWARE

La primera aproximación práctica al concepto de mecano, y al modelo de reutilización que se venía definiendo en los trabajos teóricos, se realiza mediante el desarrollo de assets y mecanos principalmente dentro del dominio del **Tratamiento Digital de Imágenes** (***TDI***), aunque no de forma exclusiva, contando con EUROWARE como soporte operativo a la reutilización. Sin embargo, para dar soporte a la estructura de mecano por parte de EUROWARE, ha sido necesario la adaptación de dicho motor de reutilización [Romay et al., 1998], [García et al., 1999]. Esta versión adaptada del repositorio se puede consultar en la dirección **http://jupiter.dcs.fi.uva.es**.

A.1.1 Adaptación de EUROWARE

Para dar soporte al concepto de Mecano sobre EUROWARE se hizo necesaria la definición en este entorno de un modelo de elemento software reutilizable que, salvando las limitaciones propias de EUROWARE, permitiera introducir en este entorno la filosofía propia de la definición de los mecanos, es decir, elementos reutilizables de grano grueso con soporte simultáneo de diferentes niveles de abstracción.

El modelo de elemento reutilizable que presenta EUROWARE no es válido para soportar directamente la estructura de mecano, principalmente porque un mecano se caracteriza por ser una agregación de assets clasificados en distintos niveles de abstracción, y esta idea no está soportada por el modelo de EUROWARE, como se puede deducir del conjunto de relaciones semánticas soportadas por dicho modelo, como se aprecia en la figura A.1.

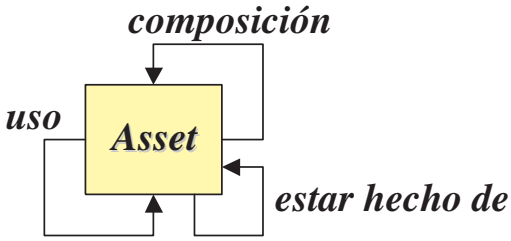

Figura A.1: Modelo de elemento reutilizable de EUROWARE.

El modelo de elemento reutilizable que se ha empleado en EUROWARE para soportar el concepto de mecano se muestra en la figura A.2. En él se utiliza una entidad denominada *Mecano* que se representa como una agregación de assets. Los assets pueden relacionarse por las relaciones intranivel agregación, composición, uso, extensión y asociación y por una relación internivel, la reificación.

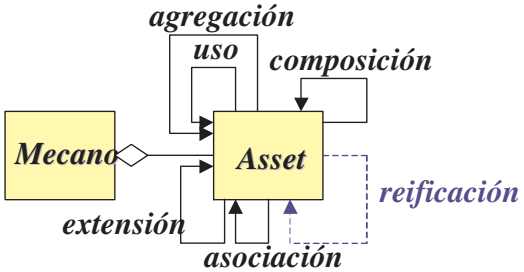

Figura A.2: Modelo de elemento reutilizable adaptado sobre EUROWARE.

La rigidez de la estructura interna de EUROWARE no lo convierten en el repositorio más adecuado para incorporar la estructura de mecano. Para salvar los problemas derivados de su

diseño interno, y poder así incorporar la estructura de mecano en EUROWARE, se ha tenido que trabajar en los siguientes aspectos:

- Ampliación del modelo del repositorio para incluir la estructura multinivel de mecano, haciéndolo de forma independiente de la estructura de asset que se mantiene.
- Incorporación de las relaciones estructurales necesarias para la definición de mecanos como elementos software reutilizables.
- Ampliación de la funcionalidad necesaria para la manipulación de los mecanos.

Para incorporar la estructura de mecano, de la forma menos costosa en cuanto a esfuerzo, se optó por mantener el modelo interno de EUROWARE, añadiéndole una nueva faceta que permite distinguir entre assets y mecanos. Estos elementos no se encuentran aislados dentro del repositorio sino enlazados mediante las dependencias entre los elementos reutilizables, las cuales se establecen mediante campos que representan a cada uno de los tipos de relaciones estructurales consideradas. Las funcionalidades propias del repositorio se mantuvieron de manera similar para los assets y los mecanos en esta primera fase. La diferencia esencial entre assets compuestos y mecanos dentro del repositorio modificado estriba en la agregación del tipo de elementos que entran a formar parte de los mecanos, esto es, los mecanos pueden estar compuestos por cualquier combinación de elementos de distinto nivel de abstracción (*con la única restricción de que estén representados al menos dos niveles de abstracción*), mientras que los assets compuestos se limitan a componentes clasificados en el mismo nivel de abstracción.

Sobre el nuevo esquema definido se tiene una estructura organizativa que permite la recuperación y el almacenamiento de cualquiera de los elementos que puede manejar el repositorio, es decir:

a) Assets simples interrelacionados entre sí por las relaciones de reificación, uso, asociación y extiende.

b) Assets complejos construidos a través de agregación de assets simples.

c) Mecanos construidos por agregación de assets simples y/o complejos, con la restricción de que exista al menos una relación de reificación entre los elementos agregados.

Las modificaciones realizadas aportan un amplio grado de libertad para la construcción de Mecanos. Así, se tienen las posibilidades que se reflejan en la figura A.3.

La figura A.3(a) representa al mecano formado por un conjunto de assets simples con la restricción de que al menos dos están clasificados en diferente nivel de abstracción y relacionados mediante una relación de reificación. La figura A.3(b) muestra un mecano como agrupación de assets compuestos, agrupados en assets de análisis, diseño e implementación con la misma restricción anterior. La figura A.3(c) presenta una mezcla de las dos anteriores, y por último la figura A.3(d) refleja un mecano como agrupación de assets simples, assets compuestos y otros mecanos.

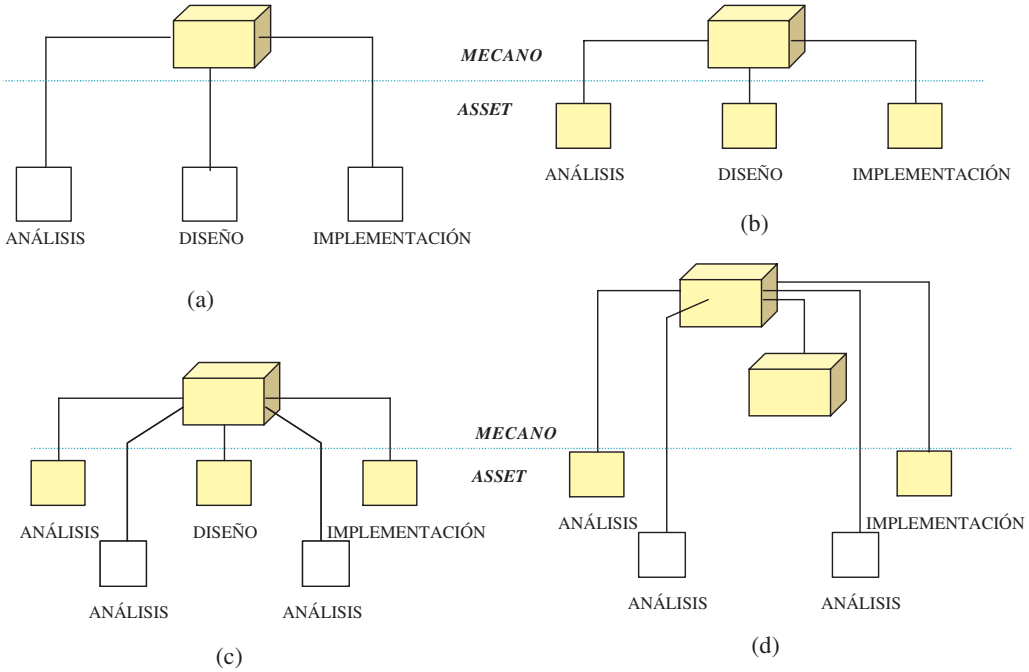

Figura A.3: Alternativas de construcción de Mecanos en EUROWARE.

En las primeras experiencias que se llevaron a cabo en la población del repositorio con mecanos se optó por la configuración exhibida por la figura A.3(b). El motivo principal de esta decisión fue facilitar la introducción y extracción de assets compuestos y mecanos, ya que los sistemas de consultas y de presentación de la información de EUROWARE no eran demasiado adecuados para el resto de las configuraciones.

A.1.2 Construcción de Mecanos en EUROWARE

El trabajo derivado de la población del repositorio, tanto con elementos de grano fino (*assets*) como con elementos de grano grueso (*mecanos*) se está llevando a cabo mediante la realización de diversos proyectos de final de carrera, tomando como áreas experimentales los dominios de aplicación de tratamiento digital de imágenes, de software para discapacitados y de gestión universitaria.

Estas actividades han dado lugar a un laboratorio de reutilización adecuado para realizar experimentos de desarrollo de aplicaciones software con reutilización y de cualificación de assets.

Para ilustrar el proceso de construcción de mecanos dentro de EUROWARE se va a presentar un ejemplo en el que se establece un mecano que representa un formato gráfico binario de una imagen, de forma que, considerando que se está trabajando bajo un paradigma objetual dirigido por responsabilidades, los componentes de este mecano serán: descripciones de requisitos funcionales y tarjetas CRC clasificados en el nivel de abstracción de análisis, diagramas de clases en el nivel

de diseño y código en el nivel de implementación.

Con el fin de facilitar las tareas de construcción y recuperación se ha adoptado una nomenclatura que permite distinguir los elementos reutilizables del repositorio de acuerdo al esquema: **Dominio-Nivel-Tipo-Nombre**.

El *dominio* hace referencia al campo de estudio donde se incluye el elemento reutilizable. Se representa mediante las iniciales de la frase que lo define. Interesa que el número de letras sea pequeño (*del orden de tres*). Si el elemento reutilizable no quiere asociarse a ningún dominio basta con dejar este campo en blanco. Dado que el ejemplo que se va a desarrollar cae dentro del dominio del tratamiento digital de imágenes, éste quedará representado por las letras TDI.

El *nivel* indica el nivel de abstracción en que se encuentra clasificado el elemento reutilizable, y puede ser ANA (*análisis*), DIS (*diseño*) o IMP (*implementación*).

El *tipo* expresa si el elemento reutilizable es un asset (A) o un Mecano (M).

Finalmente el *nombre* es un campo que se rellena libremente por el suministrador, indicando su contenido, de manera que sea descriptivo y no excesivamente grande.

Debido a las limitaciones intrínsecas de EUROWARE, el mecano se va a ir construyendo por niveles de abstracción, generando un asset complejo por cada nivel de abstracción presente en el mecano (*al menos deben existir dos por definición*) que aúne mediante agregación todos los assets del mecano clasificados en dicho nivel de abstracción.

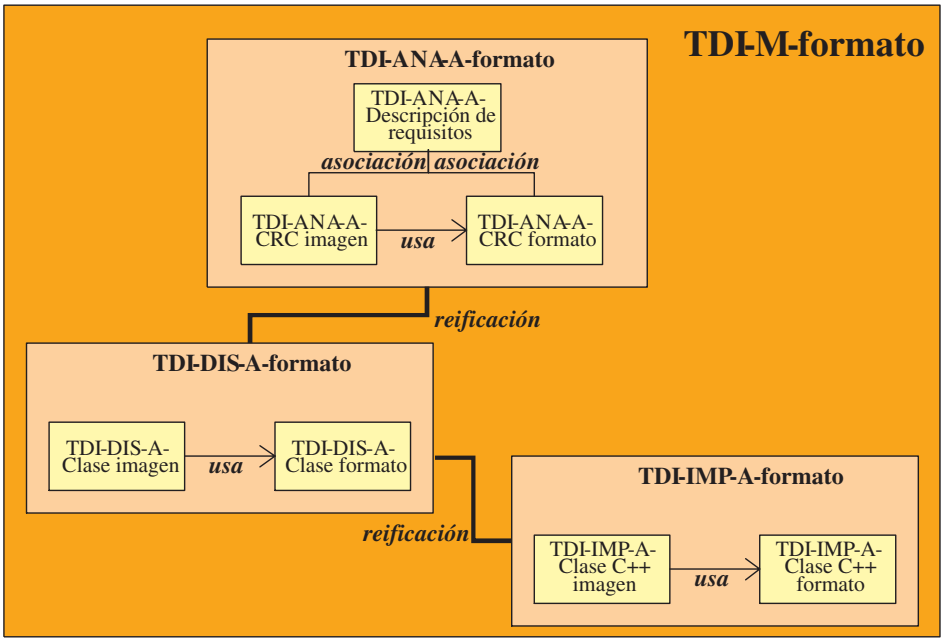

Figura A.4: Mecano TDI-M-formato.

Así, en el caso que se desarrolla se crearán tres assets complejos, el primero de ellos viene referenciado por el identificador **TDI-ANA-A-formato** y representa el nivel de análisis o especificación del formato gráfico, el segundo es **TDI-DIS-A-formato** y representa el nivel de diseño, por último el asset complejo **TDI-IMP-A-formato** representa el nivel de implementación del formato gráfico. Tal y como se puede apreciar en la figura A.4, cada uno de estos assets complejos es una agregación[1] de un conjunto de assets que están relacionados por las relaciones intranivel oportunas a cada caso.

Sin embargo, para que la estructura que se está formando pueda ser considerada como un mecano, falta introducir las relaciones internivel. En este punto, y obligados por las limitaciones de EUROWARE, se establecen relaciones de reificación entre los assets complejos que representan cada nivel de abstracción en lugar de hacerlo entre sus componentes.

Una vez que se han establecido y relacionado los assets complejos, éstos se relacionan mediante agregación con un nuevo elemento reutilizable que representa el mecano, y que viene representado por el identificador **TDI-M-formato**.

A.2 La Biblioteca de Reutilización GIRO

Tomando como base de partida las experiencias llevadas a cabo en la adaptación de EUROWARE para contar con un repositorio que soportase el concepto de mecano, así como en la población de este repositorio con assets y mecanos, se llegó a la conclusión de que EUROWARE había servido como una primera toma de contacto, pero que era del todo insuficiente, por sus propias limitaciones intrínsecas, para el soporte completo y efectivo de los trabajos que entorno a la definición de los mecanos se estaban realizando.

Así, se decidió abordar la tarea de crear una biblioteca de reutilización propia, en la que poder implementar, y experimentar, con toda flexibilidad los avances que en el terreno teórico se fueran logrando.

En la figura A.5 se presenta el esquema que originalmente sirvió de punto partida para la definición del proyecto que se pretendía abordar, y que a grandes rasgos presenta los diferentes niveles o capas con las que debía contar la biblioteca de reutilización a construir.

A continuación, en una serie de subapartados, se explica los aspectos más relevantes de la versión actual de la biblioteca de reutilización GIRO.

[1] Con el fin de facilitar la asimilación del ejemplo, en la figura A.4 se ha optado por representar la relación de agregación como contención de elementos, en lugar de utilizar una asociación con los adornos propios de algún lenguaje de modelado concreto.

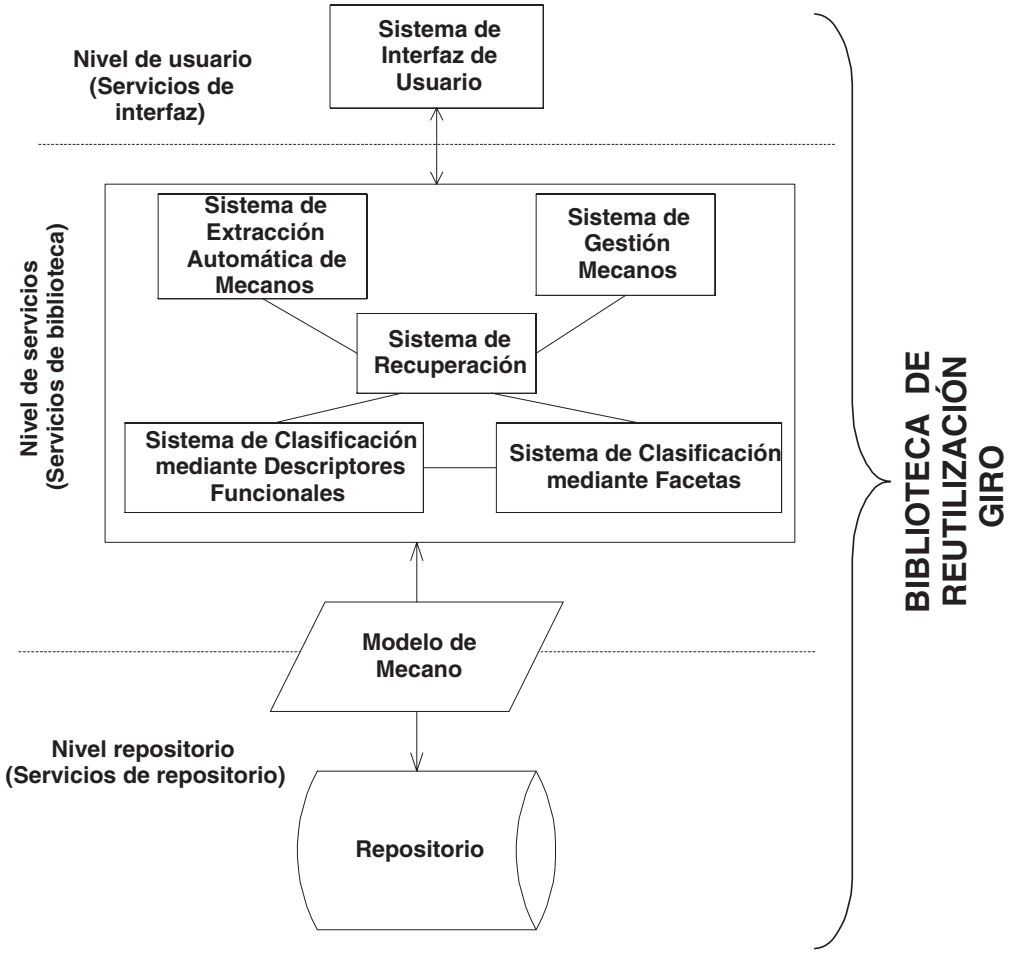

Figura A.5: Niveles presentes en la biblioteca de reutilización GIRO.

A.2.1 Propósito y Ámbito de la Biblioteca de Reutilización GIRO

El objetivo que se persigue es la creación de una biblioteca de reutilización donde los elementos reutilizables manejados sean mecanos, de forma que tanto las facilidades prestadas por la capa de repositorio y por la capa de servicios se ajusten al modelo de mecano a las reglas asociadas a dicho modelo[2].

El enfoque que se ha seguido en esta versión tiene en cuenta los dos siguientes puntos de vista:

1. *El repositorio recoge elementos reutilizables que son persistentes y deben ser almacenados y accedidos de forma fiable y eficiente.*

2. *El usuario necesita un acceso a los elementos reutilizables rápido e intuitivo, de tal manera*

[2]La primera versión de la biblioteca de reutilización se basa en el modelo de mecano contenido en [García et al., 1998c].

que se produzca una navegación a través de los elementos del repositorio de una forma sencilla mediante una interfaz gráfica.

El primer punto casa con la necesidad de que el servicio de repositorio esté basado en un soporte físico que, para el caso de esta biblioteca, se realiza sobre un sistema gestor de bases de datos objeto/relacional como es **ORACLE 8i**, para el cual se ha adaptado el modelo de mecano según se muestra en la figura A.6.

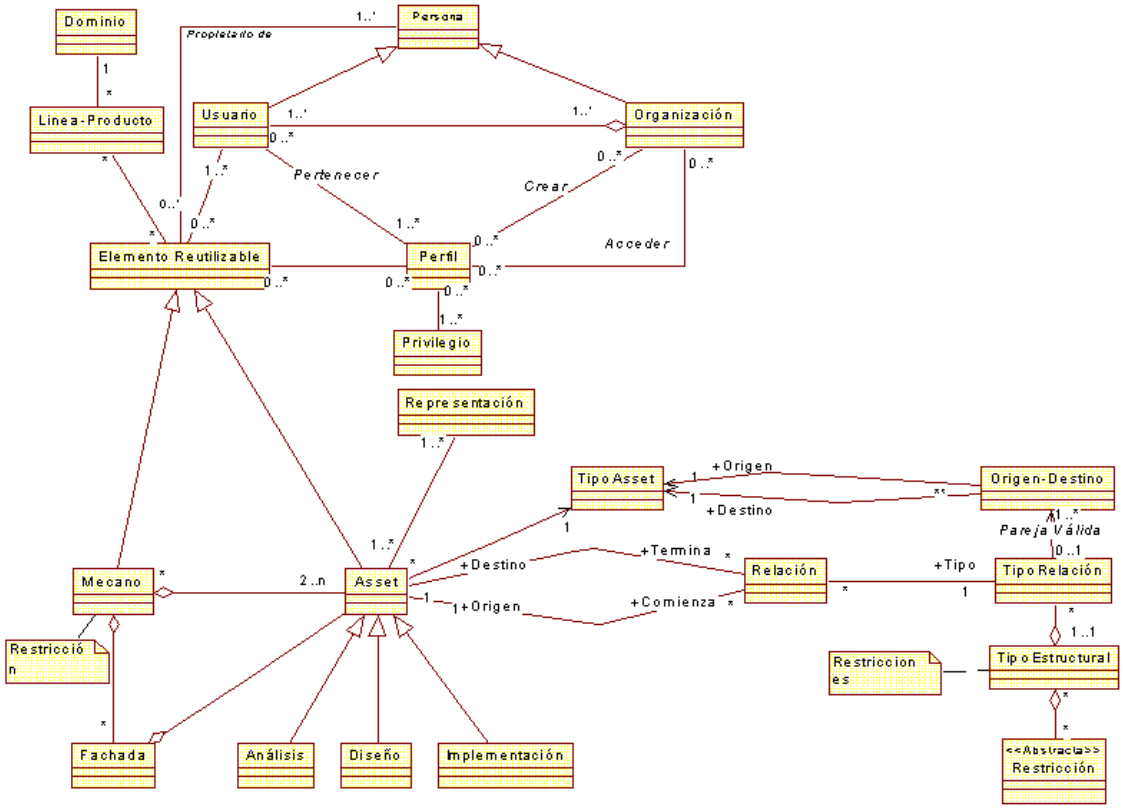

Figura A.6: Adaptación del modelo de mecano.

Para el segundo aspecto se toma la perspectiva del usuario final. En particular se asume el doble rol que puede jugar el cliente de la biblioteca: *suministrador o desarrollador con reutilización*, para la definición del conjunto de funcionalidades que se necesita soportar la biblioteca de reutilización.

En cuanto al ámbito de explotación de la biblioteca de reutilización parecía importante no vincular el acceso a la biblioteca a la situación geográfica, por lo que se estableció como un requisito esencial el acceso a través de Internet, utilizando la interfaz del navegador web como base para realizar este acceso.

La satisfacción a este requisito conduce a una arquitectura cliente/servidor, en la que se debe cuidar minimizar los recursos necesarios para efectuar el acceso y maximizar el número de posibles usuarios.

Dentro de esta arquitectura el cliente debe presentar los contenidos del repositorio de una forma visual. De forma que se exprese gráficamente toda la semántica contenida en los mecanos, sus assets y relaciones entre los assets. Para representar dichas relaciones se emplean grafos que reflejen la estructura del mecano visualmente.

Esta herramienta proporciona una nueva visión frente a las representaciones más estáticas y menos visuales, que representaban los assets sin mostrar las relaciones, como es el caso de EUROWARE. El objetivo es que el usuario no sólo introduzca u obtenga el conjunto de assets que forman un mecano, sino que pueda ver, y en definitiva comprender, como se relacionan entre sí dichos componentes.

A.2.2 La Interfaz de Usuario

La interfaz de usuario recoge los elementos básicos que necesita el usuario para interaccionar con la biblioteca de reutilización según sea el rol con el que accede a ella.

El usuario debe disponer de una visualización del grafo correspondiente al mecano que se consulta o se construye. Dicha representación debe presentar como nodos los assets que forman el mecano. Los distintos niveles de abstracción del asset así como alguna otra característica particular será representable a partir de una asignación imagen - color a cada uno de los assets. Igualmente la semántica y polaridad de las relaciones debe ser expresada gráficamente de la misma manera.

Ante esta visualización, el usuario debe tener la libertad de consultar las propiedades de los elementos de grafo de forma individual simplemente escogiendo con un apuntador, normalmente un ratón, el asset o relación del que desea conocer sus detalles. También debe ser presentada la información en una forma más textual pero que permita un rápido movimiento por los elementos del mecano, a través de visores con barras de desplazamiento.

En la figura A.7, se puede apreciar una de las múltiples ventanas que conforman la interfaz de usuario del cliente.

A.2.3 Arquitectura de la Biblioteca de Reutilización

Se han definido los requisitos por parte del usuario, siempre dando importancia vital a la representación de la información del modelo. Partiendo de que dicho modelo se almacena en un sistema objeto/relacional, queda por definir la parte de la arquitectura donde reside la parte de control de dicho modelo y desde donde se distribuyen las vistas del repositorio.

Incidiendo en el hecho de que el acceso al repositorio se realiza por medio de un navegador, parece claro que la comunicación se debe establecer a través de un servidor Web que escuche las

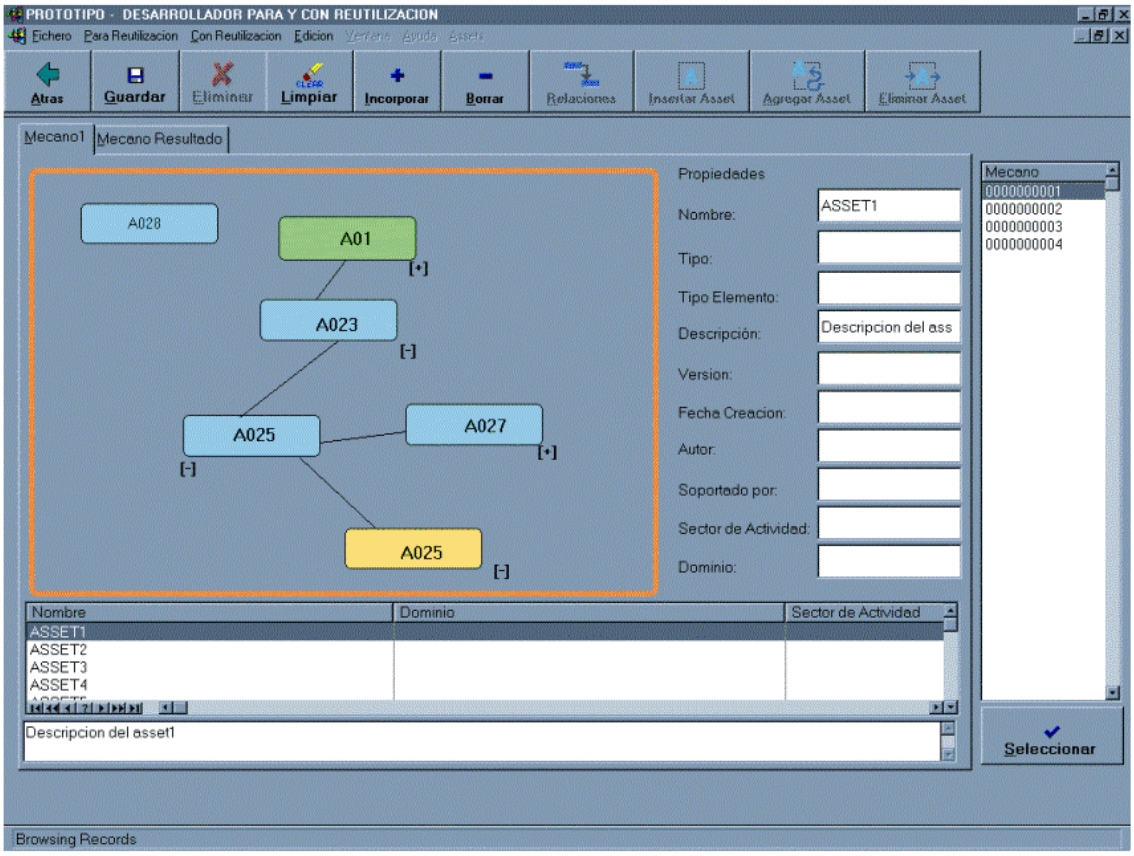

Figura A.7: Interfaz de usuario (*aplicación cliente*).

peticiones de los clientes (*navegadores*) y devuelva páginas HMTL con las vistas del modelo, como se muestra en el esquema de la figura A.8.

Para soportar los procesos que controlan el modelo se puede aprovechar el soporte de los servidores web para cargar CGIs (*Common Gateway Interface*) (*realizados en C, Perl...*) o servlets (*realizados en java*), que no son otra cosa que procesos en el lado del servidor y que generan páginas HTML hacia el cliente.

Dado que la interfaz del usuario requiere operaciones gráficas que superan las capacidades de HTML, se debe recurrir a *applets* Java que suplan las carencias gráficas de HTML. Añadido a sus facilidades gráficas, Java aporta una independencia de la plataforma, lo cual se ajusta a las pretensiones de dispersión de los clientes, sin tener por ello que realizar diferentes versiones de la aplicación cliente. Como principal desventaja se tiene que se presenta un incremento en el tamaño de las páginas HTML, y por tanto una penalización en el tiempo de distribución por la red.

ORACLE 8i aporta una nueva ventaja dentro de esta arquitectura. Una vez que el cliente carga los applets necesarios, la comunicación con el servidor se puede realizar directamente sin

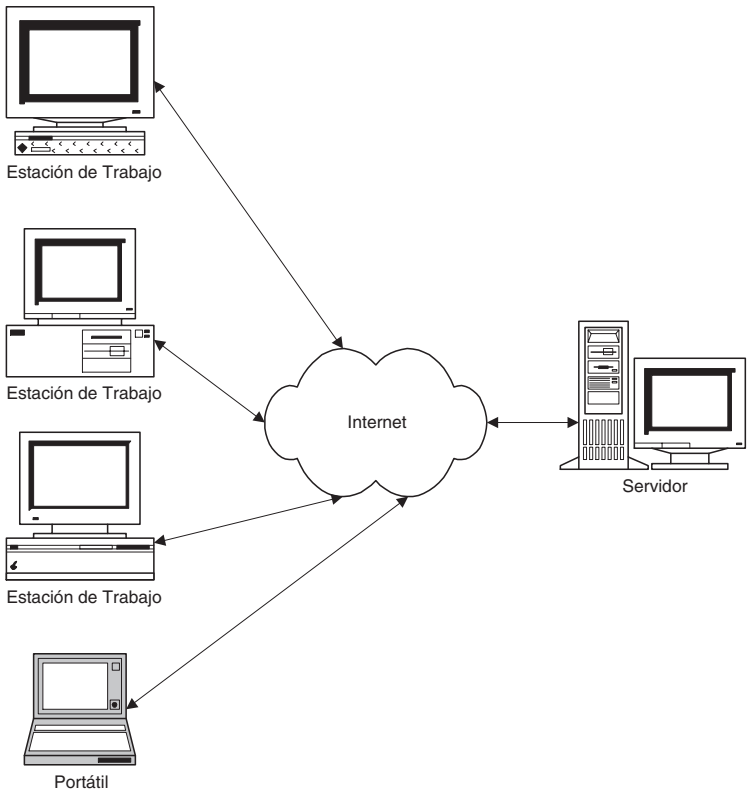

Figura A.8: Arquitectura básica.

necesidad de hacer solicitudes al servidor web. **ORACLE 8i** implementa una máquina virtual java propia (*Aurora Virtual Machine*) donde pueden residir objetos que lleven a cabo el control y acceso al repositorio (*a través de Enterprise JavaBeans u objetos CORBA*). La comunicación entre applets (*compuestos de JavaBeans*) y los Enterprise JavaBeans u objetos CORBA se realiza de forma directa y es soportada por el sistema gestor de bases de datos.

En la figura A.9 se tiene un esquema de la arquitectura cliente/servidor completa que presenta la biblioteca de reutilización GIRO.

A.3 Conclusiones y Trabajo Futuro

El aspecto práctico de esta propuesta de tesis doctoral se centra en las diversas experiencias prácticas que se han ido desarrollando en paralelo a los avances llevados a cabo en la parte teórica. Estas experiencias se pueden resumir en dos líneas: *la adaptación de un motor de repositorios existente para el soporte del concepto de mecano* y *la creación de una biblioteca de reutilización que soporte de forma nativa la noción de mecano.*

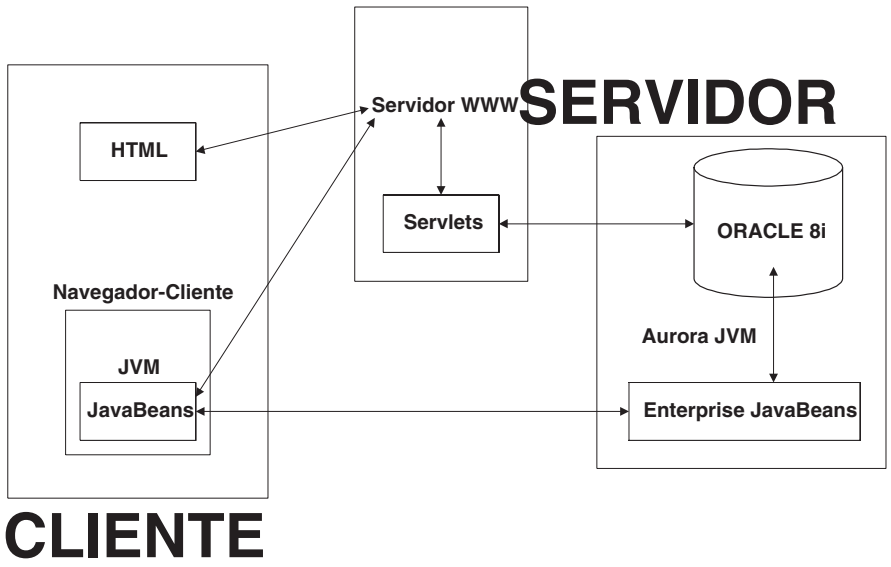

Figura A.9: Arquitectura completa.

La adaptación de EUROWARE, pese a las limitaciones de este producto, supuso una importante fuente de experiencia, especialmente en el proceso de reutilización y en la construcción de elementos reutilizables, de grano fino y grano grueso, en el apartado de desarrollo para reutilización.

Con las lecciones aprendidas, y con la convicción de que EUROWARE había cumplido su cometido, se inició la realización de una biblioteca de reutilización que soportase de forma nativa el concepto de mecano. Esta tarea, que se encuentra en la recta final de su primera fase, se ha centrado en el servicio de repositorio y en la interfaz de usuario.

El trabajo futuro se centra en terminar esta biblioteca, haciendo hincapié en implementar los procesos de construcción y extracción automática de mecanos basados en la gramática de grafos desarrollada en esta propuesta de tesis y establecer los criterios de clasificación oportunos.

Otra línea de trabajo futuro en el campo de la aplicación práctica de este proyecto de tesis pasa por transferir a la empresa el modelo de reutilización aquí propuesto.

Apéndice B

Acrónimos

ACF - Asset Certification Framework.

ACM - Association for Computing Machinery.

ADL - Architecture Description Language.

ADOO - Análisis y Diseño Orientado al Objeto.

ADT - Abstract Data Type.

AF - Applications Frames.

ALF - Asset Library Framework.

ALOAF - Asset Library Open Architecture Framework.

AOO - Análisis Orientado al Objeto.

API - Application Programming Interface.

ARC - Army Reuse Center.

ASSET - Asset Source for Software Engineering Technology.

BIDM - Basic Interoperability Data Model.

BOA - Business Object Arquitecture.

BOF - Business Object Facility.

BOM - Business Object Model.

BPR - Business Process Reengineering.

BRM - Binary Relations Model.

CACM - Communications of the ACM.

CARDS - Comprehensive Approach to Reusable Defense Software.

CARE - Computer-Aided Requirements Engineering.

CAS - Commercially Available Software.

CASE - Computer Aided/Assisted Software/System Engineering.

CBD - Component Based Development.

CBSE - Component-Based Software Engineering.

CDIF - CASE Data Interchange Format.

CDM - Common Data Model.

CFRP - Conceptual Framework for Reuse Processes.

CGI - Common Gateway Interface.

CM - Configuration Management.

CMM - Capability Maturity Model.

COM - Component Object Model.

COMMA - Common Object-oriented Methodology Metamodel Architecture.

COMN - Common Object Modelling Notation.

COP - Component-Oriented Programming.

CORBA - Common Object Request Broker Architecture.

COTS - Commercial Off-The-Shelf.

CPM - Critical Path Method.

CRC - Class, Responsability and Collaboration.

CSC - Computer Software Component.

DAG - Directed Acyclic Graph.

DARPA - Defense Advanced Research Projects Agency.

DCOM - Distributed Component Object Model.

DER - Diagrama de Entidad-Interrelación.

DFD - Diagrama de Flujo de Datos.

DLL - Dynamic-Link Libraries.

DoD - Departament of Defense.

DOO - Diseño Orientado al Objeto.

DOR - Domain Oriented Reuse.

DRI - Diccionario de Recursos de Información.

DSSA - Domain-Specific Software Architecture.

DTE - Diagrama de Transición de Estados.

ELSA - Electronic Library Services and Applicactions.

ERD - Entity-Relationship Diagram.

ERS - Especificación de Requisitos del Software.

ESA - European Space Agency.

ESI - European Software Institute.

ESPRIT - European Strategic Programme for Research and development in Information Technology.

ESSI - European Systems & Software Initiative.

EUROWARE - Enabling Users to Reuse Over Wide AREas.

FCM - Factor Criteria Metrics.

FODA - Feature-Oriented Domain Analysis.

F-ORM - Funcionality in the Objects with Roles Model.

GAF - Generic Application Frame.

GCS - Gestión de la Configuración del Software.

GEARS - Gaining Efficiency and quAlity in Real time control Software.

GIRO - Grupo de Investigación en Reutilización y Orientación al Objeto.

GoF - Gang of Four.

GOTS - Government Off-The-Shelf.

GQM - Goal-Question-Metric.

GUI - Graphics User Interface.

HLL - High Level Languages.

HP - Hewlett-Packard.

HPCC - High Performance Computation & Communication.

HTML - HyperText Markup Language.

HTTP - HyperText Transfer Protocol.

ICASE - Integrated CASE.

ICSR - International Conference on Software Reuse.

IDL - Interface Definition Language.

IEEE - The Institute of Electrical and Electronics Engineers.

IOOM - ITHACA Object-Oriented Metodology.

IPSE - Integrated Programming Support Environment.

IRD - Information Resource Dictionary.

IRDS - Information Resource Dictionary System.

IS - Ingeniería del Software.

ISC - Information System Contract.

ISO - International Standards Organization.

ITHACA - Integrated Toolkit for Highly Advanced Computers Applications.

JGL - Java Generic Library.

JOOP - Journal of Object-Oriented Programming.

JVM - Java Virtual Machine.

LaSSIE - Large Software System Information Environment.

LCM - Language for Conceptual Modeling.

LSP - Liskov Substitution Principle.

MENHIR - Modelos, Entornos y Nuevas Herramientas para la Ingeniería de Requisitos.

MIT - Masschusetts Institute of Technology.

MORE - Multimedia Oriented Repository Environment.

MRG - Modelo de Reutilización GIRO.

MVC - Model View Controller.

NASA - National Aeronautics and Space Administration.

NCSA - National Center for Supercomputing Applications.

NHSE - National HPCC Software Exchange.

NIST - National Institute of Standards and Technology.

NOC - Number of Children.

OASIS - Open and Active Specification of Information Systems

OBA - Object Behavior Analysis.

OCL - Object Constraint Language.

OCP - Open-Closed Principle.

OCX - Object Linking and Embedding Custom Controls.

ODM - Organization Domain Modeling.

ODMG - Object Database Management Group.

OLE - Object Linking and Embedding.

OM - OASIS Method.

OMA - Object Management Architecture.

OMG - Object Management Group.

OML - OPEN's Metamodel and Notation.

OMT - Object Modeling Technique.

OO - Object-Oriented.

OOA - Object-Oriented Analysis.

OOAD - Object-Oriented Analysis and Design.

OOD - Object-Oriented Design.

OOP - Object-Oriented Programming.

OOPSLA - Object-Oriented Programming Systems and LAnguages.

OOSE - Object Oriented Software Engineering.

OPEN - Object-oriented Process, Environment and Notation.

ORB - Object Request Broker.

ORM - Object with Roles Model.

OT - Object Technology.

OTAN - Organización del Tratado del Atlántico Norte.

OTUG - Object Technology Users Group.

PAL - Public Ada Library.

PCTE - Portable Common Tool Environment.

PDF - Portable Document Format.

PDL - Programs Design Lenguage.

PERT - Program Evaluation and Review Technique.

PGGC - Plan General de Garantía de Calidad.

PHS - Prosperity Heights Software.

PIE - Process Improvement Experiment.

PIER - Process Improvement Experiment in Reuse.

POO - Programación Orientada a Objetos.

RA - Requirements Analysis.

RAAT - Reuse Acquisition Action Team.

RAD - Rapid Application Development.

RBSE - Repository Based Software Engineering.

RDD - Responsability Driven Design.

REBOOT - Reuse Based on Object-Oriented Techniques.

RECAST - Requirements Composition And Specification Tool.

REP - Reuse/Release Equivalence Principle.

RFC - Request For Comments.

RFP - Request For Proposals.

RIAT - Reuse Issues Action Team.

RIB - Repository In a Box.

RIG - Reuse library Interoperability Group.

RLF - Reuse Library Framework.

RM - Reference Model.

RMFF - Reuse Methodology Fusion Framework.

ROAD - Report on Object Analysis & Design.

ROADS - Reuse Oriented Approach for Domain based Software.

ROSE - Reusable Object Software Engineering.

ROSE - Reuse Of Software Elements.

RS - Requirements Specification.

RSC - Reuse Steering Committe.

RSEB - Reuse-Driven Software Engineering Business.

RSL - Reusable Software Library.

RSRG - Reusable Software Research Group.

RUP - Rational Unified Process.

SA - Structured Analysis.

SA/SD - Structured Analysis/Structured Design.

SA-CMM - Software Acquisition Capability Maturity Model.

SAF - Specific Application Frame.

SAIC - Science Applications International Corporation.

SATC - Software Assurance Technology Center.

SDD - Software Design Document.

SDK - Software Development Kit.

SDL - Software Development Library.

SDLC - Systems Development Life Cycle.

SDM - Semantic Data Model.

SDP - Software Development Plan.

SDP - Stable Dependencies Principle.

SE - Software Engineering.

SEE - Software Engineering Environment.

SEI - Software Engineering Institute.

SEN - Software Engineering Notes.

SER - Software Evolution and Reuse.

SESC - Software Engineering Standards Committe.

SGBD - Sistema Gestor de Bases de Datos.

SGML - Standard Generalized Markup Language.

SI - Sistema de Información.

SIA - Sistema de Información Automatizado.

SIB - Software Information Base.

SLCSE - Software-Life Cycle Support Environment.

SPC - Software Productivity Centre.

SPI - Software ProcessImprovement.

SPQR - Software Productivity Quality and Reliability.

SPR - Software Productivity Research.

SQA - Software Quality Assurance.

SRBM - Software Reuse Business Model.

SRS - Software Requirements Specifcation.

SSADM - Structured System Analysis and Design Method.

SSR - Symposium on Software Reusability.

STARS - Software Technology Adaptable Reliable System.

STC - Software Technology Conference.

STL - Standard Templates Library.

STSC - Software Technology Support Center.

SURF - Software re-Use: a process impRovement experiment at an IBM Italia Facility.

TAD - Tipo Abstracto de Datos.

TDI - Tratamiento Digital de Imágenes.

TO - Tecnología de Objetos.

TOA - The Object Agency.

UE - Unión Europea.

UML - Unified Modeling Language

UPV - Universidad Politécnica de Valencia.

URL - Uniform Resource Locator.

USAL - Universidad de Salamanca.

UVA - Universidad de Valladolid.

V&V - Verificación y Validación.

VB - Visual Basic.

VBX - Visual Basic eXtensions.

VCL - Visual Component Library.

VHLL - Very High Level Languages.

W3C - World Wide Web Consortium.

WAIS - Wide Area Information Servers.

WEL - Windows Eiffel Library.

WG - Working Group.

WIMP - Windows, Icons, Menus and Pointing.

WISR - Workshop on Institutionalizing Software Reuse.

WMC - Weighted Methods per Class.

WSRD - Worldwide Software Resource Discovery.

WWW - World Wide Web.

WYSIWYG - What You See Is What You Get.

XML - eXperimental Markup Language.

YSM - Yourdon Structured Method.

Bibliografía

[Abrial, 1974] Abrial, J. R. (1974). Data semantics. In *Database Management Systems*. North Holland. J. W. Klimbie and K. L. Koffeman editors.

[Ader et al., 1990] Ader, M., Nierstrasz, O., McMahon, S., Muller, G., and Pröfrock, A.-K. (1990). The ITHACA technology: A landscape for object-oriented application development. In *Proceedings of ESPRIT'90 Conference*. Kluwer Academic Publisher.

[Aho et al., 1983] Aho, A. V., Hopcroft, J. E., and Ullman, J. D. (1983). *Data Structures and Algorithms*. Addison-Wesley.

[Alexander, 1964] Alexander, C. (1964). *Notes on the Synthesis of Form*. PhD thesis, University of Harvard. Published by Harvard University Press.

[Alexander, 1979] Alexander, C. (1979). *The Timeless Way of Building*. The Oxford University Press.

[Alexander et al., 1975] Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., and Angel, S. (1975). *The Oregon Experiment*. Oxford University Press.

[Alexander et al., 1977] Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., and Angel, S. (1977). *A Pattern Language*. Oxford University Press.

[Appleton, 1997] Appleton, B. (1997). Patterns and software: Essential concepts and terminology. *Object Magazine Online*, 3(5). Also published in http://www.enteract.com/~bradapp/docs/patterns-intro.html.

[Arango and Prieto-Díaz, 1991] Arango, G. and Prieto-Díaz, R. (1991). Part 1: Introduction and overview, domain analysis concepts and research directions. In [Prieto-Díaz and Arango, 1991], pages 9–32.

[Bandinelli and Rementería, 1996] Bandinelli, S. and Rementería, S. (1996). Software reuse introduction requires a process perspective. Technical Report ESI-1996-REUSE01, European Software Institute (ESI).

[Bandinelli and Sanz, 1997] Bandinelli, S. and Sanz, Á. (1997). The complementary aspects of process capability and reuse capability. Technical report, European Software Institute (ESI), Parque Tecnológico de Zamudio #204 E-48170 Bizkaia, Spain.

[Barros and Domínguez, 1998] Barros, J. L. and Domínguez, A., editors (1998). *Actas Del Curso Ingeniería de Software Y Reutilización: Aspectos Dinámicos Y Generación Automática*. Univeridade De Vigo, Escuela Universitaria de Ingeniería Técnica en Informática de Gestión. Edificio Politécnico - Ourense. Ourense, del 6 al 10 de Julio de 1998.

[Basili et al., 1999] Basili, V., Baxter, I., Griss, M. L., Karlsson, E.-A., and Perry, D. (1999). Reuse R&D: Gap between theory and practice. In *Proceedings of the Fifth Symposium on Software Reusability SSR'99*, pages 172–177, May 21-23, 1999, Los Angeles, CA (USA).

[Basili, 1994] Basili, V. R. (1994). Facts and myths affecting software reuse. In *Proceedings of the 16th International Conference on Software Engineering (ICSE'94)*, page 269, May 16-21, 1994, Sorrento (Italy). IEEE Computer Society.

[Bass et al., 1999] Bass, L., Campbell, G., Clements, P. C., Nothrop, L., and Smith, D. (1999). Third product line practice workshop report. Technical Report CMU/SEI-99-TR-003 (ESC-TR-99-003), Software Engineering Institute, Pittsburgh, PA 15213 (USA).

[Bass et al., 1998] Bass, L., Chastek, G., Clements, P. C., Northrop, L., Smith, D., and Withey, J. (1998). Second product line practice workshop report. Technical Report CMU/SEI-98-TR-015 (ESC-TR-98-015), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA).

[Bass et al., 1997] Bass, L., Clements, P. C., Cohen, S., Northrop, L., and Withey, J. (1997). Product line practice workshop report. Technical Report CMU/SEI-97-TR-003 (ESC-TR-97-003), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA).

[Batory and Poulin, 1999] Batory, D. and Poulin, J., editors (1999). *Proceedings of the 9th Workshop on Institutionalizing Software Reuse (WISR'9)*, Thompson Conference Center. The University of Texas at Austion (USA). January 7-9, 1999.

[Batory et al., 1994] Batory, D., Singhal, V., Thomas, J., Dasar, S., Geraci, B., and Sirkin, M. (1994). The GenVoca model of software system generators. *IEEE Software*, 11(9):89–94.

[Bellinzona et al., 1993] Bellinzona, R., Fugini, M. G., and de Mey, V. (1993). Reuse of specifications and designs in a development information system. In Prakash, N., Rolland, C., and Percini, B., editors, *Information System Development Process*, pages 79–96, Amsterdam. North-Holland.

[Bellinzona et al., 1995] Bellinzona, R., Fugini, M. G., and Pernici, B. (1995). Reusing specifications in OO applications. *IEEE Software*, 12(2):65–75.

[Bergey et al., 1998] Bergey, J., Clements, P. C., Cohen, S., Donohoe, P., Jones, L., Krut, B., Northrop, L., Tilley, S., Smith, D., and Withey, J. (1998). DoD product line practice workshop report. Technical Report CMU/SEI-98-TR-007 (ESC-TR-98-007), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA).

[Bernstein and Dayal, 1994] Bernstein, P. A. and Dayal, U. (1994). An overview of repository technology. In *The Proceedings of the 20th International Conference on Very Large Data Bases*, pages 705–713, Santiago - Chile.

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

[Biddle and Tempero, 1998] Biddle, R. L. and Tempero, E. D. (1998). Towards tool support for reuse. In *Proceedings of the 1998 International on Software Engineering: Education & Practice (SEEP'98)*, pages 126–133, 26-29 January, 1998. Dunedin, New Zeland. IEEE Computer Society.

[Biggerstaff, 1989] Biggerstaff, T. J. (1989). Design recovery for maintenance and reuse. *IEEE Computer*, 22(7):36–49.

[Biggerstaff, 1992] Biggerstaff, T. J. (1992). An assessment and analysis of software reuse. *Advances in Computers*, 34:1–57.

[Biggerstaff, 1999] Biggerstaff, T. J. (1999). Reuse technologies and their niches. In *Proceedings of the 1999 International Conference on Software Engineering (ICSE' 99)*, pages 613–614, May 16-22, 1999, Los Angeles, CA (USA). ACM.

[Biggerstaff and Perlis, 1989] Biggerstaff, T. J. and Perlis, A. J., editors (1989). *Software Reusability. Concepts and Models*, volume I of *Frontier Series*. ACM Press. Addison Wesley, New York.

[Biggerstaff and Richter, 1987] Biggerstaff, T. J. and Richter, C. (1987). Reusability framework, assement and directions. *IEEE Software*, 4(2):41–49.

[Blaha and Premerlani, 1998] Blaha, M. and Premerlani, W. (1998). *Object-Oriented Modeling and Design for Database Applications*. Prentice-Hall, Upper Saddle River, New Jersey 07458.

[Boehm and Abts, 1999] Boehm, B. and Abts, C. (1999). COTS integration: Plug and pray? *Computer*, 32(1):135–138.

[Booch, 1987] Booch, G. (1987). *Software Components with Ada - Structures, Tools and Subsystems*. Benjamin Cummings.

[Booch, 1994] Booch, G. (1994). *Object-Oriented Analysis and Design, with Applications*. The Benjamin/Cummins Publishing Company, Redwood City, CA, 2nd edition.

[Booch et al., 1999] Booch, G., Rumbaugh, J., and Jacobson, I. (1999). *The Unified Modeling Language User Guide*. Object Technology Series. Addison-Wesley.

[Bracchi et al., 1976] Bracchi, G., Paolini, P., and Pelagatti, G. (1976). Binary logical associations in data modelling. In *Modelling in Data Base Management Systems*. North-Holland Publishing. G.M. Nijssen editor.

[Bray et al., 1998] Bray, T., Paoli, J., and Sperberg-McQueen, C. M. (1998). Extensible markup language (XML) 1.0. W3C Recommendation 10-Feb-98 REC-xml-19980210, W3C. http://www.w3.org/TR/1998/REC-xml-19980210.pdf.

[Brooks, 1987] Brooks, F. (1987). No silver bullet: Essence and accident of software engineering. *IEEE Software*, 20(4):12.

[Brown and Wallnau, 1998] Brown, A. W. and Wallnau, K. C. (1998). The current state of CBSE. *IEEE Software*, 15(5):37–46.

[Browne et al., 1995] Browne, S. V., Dongarra, J., Green, S., Moore, K., Rowan, T., Wade, R., Fox, G., Hawick, K., Kennedy, K., Pool, J., Stevens, R., Olson, R., and Disz, T. (1995). The national HPCC software exchange. *IEEE Computational Science and Engineering*, 2(2):62–69.

[Browne and Moore, 1997] Browne, S. V. and Moore, J. W. (1997). Reuse library interoperability and the World Wide Web. In *Proceedings of the 19th Internacional Conference on Software Engineering (ICSE 97)*, pages 684–691, May 17-23, 1997. Boston, Massachusetts (USA). IEEE.

[Brownsword and Clements, 1996] Brownsword, L. and Clements, P. C. (1996). A case study in successful product line development. Technical Report CMU/SEI-96-TR-016 (ESC-TR-96-016), Software Engineering Institute (SEI) - Carnegie Mellon University, Pittsburgh, Pennsylvania 15213 (USA). Product Line Systems Program.

[Buxton et al., 1976] Buxton, J. M., Naur, P., and Randell, B., editors (1976). *Software Engineering Concepts and Techniques; 1968 NATO Conference on Software Engineering*. Van Nostrand Reinhold.

[Carbone and Araneo, 1999] Carbone, F. and Araneo, A. (1999). Gaining efficiency and quality in real time control software. Process Improvement Experiment Final Report ESSI Project 24189, European Systems & Software Initiative (ESSI). Version 5.0.

[Carney, 1997] Carney, D. (1997). Assembling large systems from COTS components: Opportunities, cautions, and complexities. Sei monographs on use of commercial software in government systems, Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA).

[Cattell and Barry, 1997] Cattell, R. and Barry, D., editors (1997). *The Object Database Standard: ODMG 2.0*. Morgan Kaufmann.

[Cheatham, 1984] Cheatham, T. E. J. (1984). Reusability through program transformations. *IEEE Transactions on Software Engineering*, SE-10(5):589–594.

[Christel and Kang, 1992] Christel, M. G. and Kang, K. C. (1992). Issues in requirements elicitation. Technical Report CMU/SEI-92-TR-12, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213.

[Clark and Porter, 1997] Clark, P. and Porter, B. (1997). Building concept representations from reusable components. In *Proceedings of the AAAI'97*, pages 369–376.

[Clements, 1997a] Clements, P. C. (1997a). Report of the reuse and product lines working group of WISR8. Special Report CMU/SEI-97-SR-010, Software Engineering Institute. Carnegie Mellon University, Pittsburgh, Pennsylvania 15213 (USA).

[Clements, 1997b] Clements, P. C. (1997b). Successful product line engineering requires more than reuse. In *Proceedings of 8th Annual Workshop on Software Reuse WISR-8*, March 23-26, 1997. The Ohio State University, Columbus, Ohio (USA).

[Clements et al., 1998] Clements, P. C., Northrop, L. M., Bachmann, F., Bass, L., Bergey, J., Chastek, G., Cohen, S., Donohoe, P., Jones, L., Krut, R., Little, R., Smith, D., Tilley, S., Weiderman, N., Withey, J., and Woods, S. (1998). A framework for software product line practice - version 1.0. Product line systems program, Software Engineering Institute (SEI) - Carnegie Mellon University, Pittsburgh, PA 15213 (USA).

[Cohen et al., 1996] Cohen, S., Friedman, S., Martin, L., Royer, T., Solderitsch, N., and Webster, R. (1996). Concept of operations for the ESC product line approach. Technical Report CMU/SEI-96-TR018 (ESC-TR-96-18), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, Pennsylvania 15213 (USA).

[Cohen et al., 1995] Cohen, S., Friedman, S., Martin, L., Solderitsch, N., and Webster, R. (1995). Product line identification for ESC-Hanscom. Special Report CMU/SEI-95-SR-024, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213 (USA).

[Constantopoulos and Dörr, 1995] Constantopoulos, P. and Dörr, M. (1995). Component classification in the software information base. In Nierstrasz, O. and Tsichritzis, D., editors, *Object-Oriented Software Composition*, pages 177–200. Prentice-Hall.

[Constantopoulos et al., 1992] Constantopoulos, P., Jarke, M., Mylopoulos, J., and Vassiliou, Y. (1992). The software information base: A server for reuse. Technical Report ITHACA.FORTH.92.E2.#1, FORTH Computer Science Institute, Iraklion (Greece).

[Coplien, 1998] Coplien, J. O. (1998). A pattern definition - software patterns. http://hillside.net/patterns/definition.html.

[Creps et al., 1992] Creps, R. E., Simos, M. A., and Prieto-Díaz, R. (1992). The STARS conceptual framework for reuse processes. In *Proceedings of STARS'92*.

[Cybulski, 1996] Cybulski, J. L. (1996). Introduction to software reuse. Technical Report TR 96/4, Department of Information Systems. University of Melbourne.

[Cybulski, 1997] Cybulski, J. L. (1997). Reuse of early life-cycle artefacts: Reusing requirements with word processor? In *Proceedings of the 8th Annual Workshop on Software Reuse WISR-8*, March 23-26, 1997. The Ohio State University, Columbus, Ohio (USA).

[Cybulski et al., 1997a] Cybulski, J. L., Neal, R. D., Kram, A., and Allen, J. C. (1997a). Characterisation of the early life-cycle artefacts: Summary of findings from WISR-8/working group #5. http://mate.dis.unimelb.edu.au/jacob/publications/wisr-8/WG5Report.htm. WISR-8: 8th Annual Workshop on Software Reuse/Working Group #5.

[Cybulski et al., 1997b] Cybulski, J. L., Neal, R. D., Kram, A., and Allen, J. C. (1997b). Report on the reuse of early life-cycle artefacts. http://mate.dis.unimelb.edu.au/jacob/publications/wisr-8/WG5Brief.htm. WISR-8: 8th Annual Workshop on Software Reuse/Working Group #5.

[de Mey and Nierstrasz, 1993] de Mey, V. and Nierstrasz, O. (1993). The ITHACA application development environment. *Visual Objects*, pages 267–280. ed. Tsichritzis, Centre Universitaire d'Informatique, University of Geneva (Switzerland).

[Dean and Cordy, 1995] Dean, T. R. and Cordy, J. R. (1995). A syntactic theory of software architecture. *IEEE Transactions on Software Engineering*, 21(4):302–313.

[Degano and Montanari, 1987] Degano, P. and Montanari, U. (1987). A model for distributed systems based on graph rewriting. *Journal of the Association for Computing Machinery (ACM)*, 34(2):411–449.

[Denker, 1995] Denker, G. (1995). Transactions in object-oriented specifications. In Astesiano, E., Reggio, G., and Tarlecki, A., 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*, Berlin, LNCS 906. Springer.

[Denker, 1996] Denker, G. (1996). Reification - changing viewpoint but preserving truth. In Haveraan, M., Owe, O., and Dahl, O., 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.

[Denker and Ehrich, 1995] Denker, G. and Ehrich, H.-D. (1995). Action reification in object oriented specification. In Wieringa, R. J. and Feenstra, R. B., editors, *Information Systems - Correctness and Reusability, Selected Papers from the IS-CORE Workshop*, pages 103–118. World Scientific.

[Do Prado Leite et al., 1994] Do Prado Leite, J. C. S., Sant'Anna, M., and de Freitas, F. G. (1994). Draco-PUC: A technology assembly for domain oriented software development. In Frakes, W. B., editor, *Proceedings of the Third International Conference on Software Reusability ICSR-3*, pages 102–109, Rio de Janeiro, Brazil, 1-4 November 1994. IEEE Press.

[DoD, 1988] DoD (1988). Military standard: Defense system software development. Military Standard MIL-STD-2167A, Department of Defense, Washington, DC.

[DoD, 1992] DoD (1992). DoD software reuse vision and strategy. Technical Report 1222-04-210/40, Department of Defense (DoD Software Reuse Initiative), Falls Church, VA.

[DoD, 1995] DoD (1995). *Technology Roadmap Version 2.2*, volume 1: Technology Assessment. Department of Defense (DoD Software Reuse Initiative), Falls Church, VA.

[DoD, 1996] DoD (1996). Software reuse executive primer. Produced by DOD Software Reuse Initiative.

[D'Souza and Wills, 1999] D'Souza, D. F. and Wills, A. C. (1999). *Objects, Components and Frameworks with UML. The Catalysis Approach.* Object Technology Series. Addison Wesley.

[Dubinsky et al., 1989] Dubinsky, E., Freudenberger, S., Schonberg, E., and Schwartz, J. T. (1989). Reusability of design for large software systems: An experiment with the SETL optimizer. In [Biggerstaff and Perlis, 1989], pages 275–293.

[Durán and Bernárdez, 1998] Durán, A. and Bernárdez, B. (1998). Norma para la recolección de requisitos de un sistema software. Technical Report Versión 1.1, Departamento de Lenguajes y Sistemas Informáticos. Universidad de Sevilla.

[Durán and Bernárdez, 1999] Durán, A. and Bernárdez, B. (1999). Metodología para la elicitación de requisitos de sistemas software. Informe Técnico Versión 2.0, Departamento de Lenguajes y Sistemas Informáticos. Universidad de Sevilla.

[Durnin et al., 1996] Durnin, M. A., Terry, K., and Sullins, R. (1996). Establishing a repository for enterprise wide software reuse. In *Proceedings of the Fifth Annual Workshop on Software Reuse Education and Training - Reuse'96*. http://www.asset.com/WSRD/conferences/proceedings/papers/sullins/mnpaper2.htm.

[Edwards et al., 1990] Edwards, S. H., Gibson, D. S., Weide, B. W., and Zhupanov, S. (1990). The 3C model of reusable software components. In *Proceedings of the Third Annual Workshop: Methods and Tools for Reuse*.

[Edwards et al., 1997] Edwards, S. H., Gibson, D. S., Weide, B. W., and Zhupanov, S. (1997). Software component relationships. In *Proceedings of 8th Annual Workshop on Software Reuse WISR-8*, March 23-26, 1997. The Ohio State University, Columbus, Ohio (USA).

[Eichmann, 1995] Eichmann, D. (1995). The repository based software engineering program. In *Proceedings of the Fifth Systems Reengineering Technology Workshop*, Monterrey, CA. February 7-9.

[Eichmann et al., 1995] Eichmann, D., Price, M., Terry, R. H., and Welton, L. L. (1995). ELSA and MORE: A library and an environment for the web. http://rbse.mountain.net/MOREplus/ELSAandMORE.

[Favaro, 1999] Favaro, J. (1999). Strategic analysis of component-based development. In [Batory and Poulin, 1999]. http://www.umcs.maine.edu/~ ftp/wisr/wisr9/final-papers/Favaro.html.

[Fenton and Hill, 1993] Fenton, N. and Hill, G. (1993). *Systems Constructions and Analysis: A Mathematical and Logical Framework*. McGraw-Hill International.

[Firesmith et al., 1996] Firesmith, D., Henderson-Sellers, B., Graham, I., and Page-Jones, M. (1996). *OPEN Modeling Language (OML) Reference Manual*. OPEN Consortium. Version 1.0.

[Fischer, 1987] Fischer, G. (1987). Cognitive view of reuse and redesign. *IEEE Software*, 4(4):60–72.

[Frakes and Isoda, 1994] Frakes, W. and Isoda, S. (1994). Success factors of systematic reuse. *IEEE Software*, 11(5):15–18.

[Frakes and Terry, 1996] Frakes, W. and Terry, C. (1996). Software reuse: Metrics and models. *ACM Computing Surveys*, 28(2):415–435.

[Frakes et al., 1991] Frakes, W. B., Biggerstaff, T. J., Matsumura, K., Prieto-Díaz, R., and Schaefer, W. (1991). Software reuse: Is it delivering? In *Proceedings of the 13th International Conference on Software Engineering (ICSE'91)*, pages 52–59, May 13-17, 1991, Austin, TX (USA). IEEE Computer Society.

[Frakes and Fox, 1995] Frakes, W. B. and Fox, C. J. (1995). Sixteen questions about software reuse. *Communications of the ACM*, 38(6):75–87.

[Freeman, 1987a] Freeman, P. (1987a). A perspective on reusability. In [Freeman, 1987c], pages 2–8.

[Freeman, 1987b] Freeman, P. (1987b). Reusable software engineering: Concepts and research directions. In [Freeman, 1987c], pages 10–23.

[Freeman, 1987c] Freeman, P., editor (1987c). *Tutorial: Software Reusability*, Washington, D. C. IEEE Computer Society Press.

[Fugini et al., 1992] Fugini, M. G., Nierstrasz, O., and Pernici, B. (1992). Application development through reuse: The ITHACA tools environment. *ACM SIGOIS Bulletin*, 13(2):38–47.

[Gamma et al., 1993] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1993). Design patterns: Abstraction and reuse of object-oriented design. In Nierstrasz, O. M., editor, *Proceeding of the 7th European Conference in Object Oriented Programming - ECOOP'93*, pages 406–431. Springer-Verlag.

[Gamma et al., 1995] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley.

[García, 1998] García, F. J. (1998). Patrones. De Alexander a la tecnología de objetos. *RPP - Revista Profesional para Programadores*, V(45):44–52. http://olivo.usal.es/~ fgarcia/doc/patrones1.pdf.

[García and Marqués, 1998] García, F. J. and Marqués, J. M. (1998). El principio de sustitución de Liskov. *RPP - Revista Profesional para Programadores*, V(42):40–44.

[García and Marqués, 1999] García, F. J. and Marqués, J. M., editors (1999). *Actas de Las IV Jornadas de Trabajo MENHIR*, May 6-7, 1999. Sedano (Burgos) - Spain. Organized by GIRO (Grupo de Investigación en Reutilización y Orientación a Objeto) - Valladolid University (Spain).

[García et al., 1998a] García, F. J., Marqués, J. M., Laguna, M. Á., and Maudes, J. M. (1998a). Estructuras complejas de reutilización: Definición de mecano estático. In Carsí, J. Á., editor, *Actas de Las II Jornadas de Trabajo MENHIR*, pages 135–141, 19-20 de Febrero, 1998. Valencia (España). Organizado por el Departamento de Sistemas Informáticos y Computación de la Universidad Politécnica de Valencia.

[García et al., 1998b] García, F. J., Marqués, J. M., Laguna, M. Á., and Maudes, J. M. (1998b). Influencia de las relaciones entre elementos software reutilizables en la generación de mecanos. In Toval, A. and Nicolás, J., editors, *Actas de Las III Jornadas En Ingenería Del Software (JIS'98)*, pages 155–166, Murcia, 11-13 de Noviembre de 1998.

[García et al., 1998c] García, F. J., Marqués, J. M., Laguna, M. Á., and Maudes, J. M. (1998c). Mecanos: Soporte de diferentes niveles de abstracción en los elementos software reutilizables. Technical Report TR-GIRO-02-98V1.2.1, Departamento de Informática, Edificio de Tecnologías

de la Información y las Telecomunicaciones. Universidad de Valladolid. Campus Miguel Delibes, 47011 Valladolid (España).

[García et al., 1997a] García, F. J., Marqués, J. M., and Maudes, J. M. (1997a). Análisis y diseño orientado al objeto para reutilización. Technical Report TR-GIRO-01-97V2.1.1, Universidad de Valladolid. Versión 2.1.1.

[García et al., 1997b] García, F. J., Marqués, J. M., and Maudes, J. M. (1997b). Mecano: Una propuesta de componente software reutilizable. In Díaz, O. and Lopistéguy, P., editors, *Actas de Las II Jornadas de Ingeniería Del Software*, pages 232–244, Donostia-San Sebastián, 3-5 de Septiembre de 1997.

[García et al., 1998d] García, F. J., Marqués, J. M., and Maudes, J. M. (1998d). Mecanos as basis of a Compositional/Generative mixed reuse model. In *Proceedings of the 2nd Edition of the European Reuse Workshop (ERW'98)*, pages 17–20 (Volumen 2), November, 4-6, 1998. Madrid (Spain). European Software Institute (ESI).

[García et al., 1999] García, F. J., Romay, M. P., Marqués, J., and Crespo, Y. (1999). Mecanos: Exposición de resultados y líneas de trabajo abiertas en la reutilización sistemática del software. In Pastor, O., González, C., Trejos, I., and Insfrán, E., editors, *Actas de Las II Jornadas Iberoamericanas de la Ingeniería de Requisitos Y Ambientes de Software (IDEAS'99)*, pages 193–204, 24-26 de Marzo 1999, Alajuela (Costa Rica).

[Girardi, 1992] Girardi, M. R. (1992). Application engineering: Putting reuse to work. In Tsichritzis, D., editor, *Object Frameworks*, pages 137–149. Centre Universitaire d'Informatique, University of Geneva.

[Girardi, 1998] Girardi, M. R. (1998). Main approaches to software classification and retrieval. In [Barros and Domínguez, 1998]. Ourense, del 6 al 10 de Julio de 1998.

[Girow, 1996a] Girow, A. (1996a). Binary relations approach to building object database model. *Object Currents*, 1(11).

[Girow, 1996b] Girow, A. (1996b). Objects and binary relations. *Object Currents*, 1(6).

[Glass, 1998] Glass, R. L. (1998). Reuse: What's wrong with this picture? *IEEE Software*, 15(2):57–59.

[Gomaa, 1995] Gomaa, H. (1995). Reusable software requirements and architectures for families of systems. *The Journal of Systems Software*, 28:189–202.

[Gotel and Finkelstein, 1994] Gotel, O. C. Z. and Finkelstein, A. C. W. (1994). An analysis of the requirements traceability problem. In *Proceedings of The First International Conference on Requirements Engineering*, Colorado Springs.

[Gregory, 1995] Gregory, P. (1995). CARDS support for systematic reuse technology transfer. *CrossTalk*, 8(7).

[Griss, 1991] Griss, M. L. (1991). Bus-based kits for reusable software. In *Proceedings of the Fourth Workshop on Institutionalizing Software Reuse WISR-4*. ftp://gandalf.umcs.maine.edu/pub/WISR/wisr4/proceedings/ps/griss.ps.

[Griss, 1993a] Griss, M. L. (1993a). Software reuse: From library to factory. *IBM System Journal*, 32(4):1–23.

[Griss, 1993b] Griss, M. L. (1993b). Towards tools and languages for hybrid domain-specific kits. In Paulin, J. and Tracz, W., editors, *Proceedings of the Sixth Workshop on Institutionalizing Software Reuse (WISR-6)*. ftp://gandalf.umcs.maine.edu/pub/WISR/wisr6/proceedings/ps/griss.ps.

[Griss, 1995a] Griss, M. L. (1995a). The architecture and processses for systematic OO reuse factory. In *Proceedings of the 7th Workshop on Institutionalizing Software Reuse (WISR-7)*, Andersen Consulting's Center for Professional Education, St. Charles, Illinois, 28-30 August 1995.

[Griss, 1995b] Griss, M. L. (1995b). Packaging software reuse technologies as kits. *Object Magazine*, 5(6):80–81.

[Griss, 1995c] Griss, M. L. (1995c). Software reuse: A process of getting organized. *Object Magazine*, 4(12). http://www.hpl.hp.com/reuse/papers/obm2.htm.

[Griss, 1995d] Griss, M. L. (1995d). Software reuse: Objects and frameworks are not enough. *Object Magazine*, 4(9):77–79. http://www.hpl.hp.com/reuse/papers/obm1.htm.

[Griss, 1996a] Griss, M. L. (1996a). Domain engineering and variability in the reuse-driven software engineering business. *Object Magazine*, 6(7). http://www.hpl.hp.com/reuse/papers/obm7.htm.

[Griss, 1996b] Griss, M. L. (1996b). Systematic OO software reuse - a year of progress. *Object Magazine*, 5(9):82–83. http://www.hpl.hp.com/reuse/papers/obm5.htm.

[Griss, 1996c] Griss, M. L. (1996c). Systematic software reuse: Architecture, process and organization are crucial. *Fusion Newsletter*. http://www.hpl.hp.com/reuse/papers/fusion1.htm.

[Griss, 1999a] Griss, M. L. (1999a). Architecting for large-scale systematic component reuse. In *Proceedings of the 1999 International Conference on Software Engineering (ICSE 99)*, pages 615–616, May 16-22, 1999. Los Angeles, CA (USA).

[Griss, 1999b] Griss, M. L. (1999b). Reuse 2001-2004. what next, now that we have solved all reuse problems? In [Batory and Poulin, 1999]. http://www.umcs.maine.edu/~ ftp/wisr/wisr9/final-papers/Griss.html.

[Griss et al., 1997] Griss, M. L., Favaro, J., and D'Alessandro, M. (1997). Developing architecture through reuse. *Object Magazine*, 7(4):35–41.

[Griss et al., 1998] Griss, M. L., Favaro, J., and D'Alessandro, M. (1998). Integrating feature modeling with RSEB. In *Proceedings of the Fifth International Conference on Software Reuse (ICSR-5)*, pages 76–85, 2-5 June, 1998. Victoria, B. C., Canada. IEEE Computer Society, IEEE Press.

[Griss and Kessler, 1996] Griss, M. L. and Kessler, R. R. (1996). Building object-oriented instrument kits. *Object Magazine*, 5(11):71–81.

[Griss and Wentzel, 1993] Griss, M. L. and Wentzel, K. D. (1993). Hybrid reuse with domain-specific kits. In Paulin, J. and Tracz, W., editors, *WISR'93:6th Annual Workshop on Software Reuse. Summary and Working Group Reports.*

[Griss and Wentzel, 1994] Griss, M. L. and Wentzel, K. D. (1994). Hybrid domain-specific kits for a flexible software factory. In *Proceedings of SAC'94, Reuse and Reengineering Track*, pages 47–52, New York. ACM.

[Griss and Wentzel, 1995] Griss, M. L. and Wentzel, K. D. (1995). Hybrid domain-specific kits. *The Journal of Systems and Software*, 30(3):213–230. Special Issue on Software Reusability.

[Griss and Wosser, 1995] Griss, M. L. and Wosser, M. (1995). Making reuse work at Hewlett-Packard. *IEEE Software*, 12(1):105–107.

[Grupo MENHIR, 1998] Grupo MENHIR (1998). MENHIR (Modelos, Entornos y Nuevas Herramientas para la Ingeniería de Requisitos). In Casamayor, J. C., Celma, M., Mota, L., and Pastor, M. Á., editors, *Actas de Las III Jorndas de Investigación Y Docencia En Bases de Datos*, pages 11–41, Valencia (España), 23 de Marzo de 1998. Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia, Servicio de Publicaciones de la Universidad Politécnica de Valencia (SPUPV-98.2136).

[Guerrieri, 1999] Guerrieri, E. (1999). Reuse sucess - when and how? In [Batory and Poulin, 1999]. http://www.umcs.maine.edu/~ ftp/wisr/wisr9/final-papers/Guerrieri.html.

[Haase, 1996] Haase, K. (1996). *Invention and Exploration in Discovery.* PhD thesis, MIT Media Laboratory.

[Henderson-Sellers, 1997] Henderson-Sellers, B. (1997). OPEN relationships - compositions and containments. *Journal of Object-Oriented Programming (JOOP)*, pages 51–55.

[Henderson-Sellers and Edwards, 1994] Henderson-Sellers, B. and Edwards, J. M. (1994). MOSES: a second generation object-oriented methodology. *Object Magazine*, 4(6):68–73.

[Henderson-Sellers and Pant, 1993] Henderson-Sellers, B. and Pant, Y. (1993). Adopting the reuse mindset throughout the lifecycle. *Object Magazine*, 3(4):73–75.

[Henry and Faller, 1995] Henry, E. and Faller, B. (1995). Large-scale industrial reuse to reduce cost and cycle time. *IEEE Software*, 12(5):47–53.

[Hirsch et al., 1998] Hirsch, D., Inverardi, P., and Montanari, U. (1998). Graph grammars and constraint solving for software architectures styles. In *Proceedings of the Third International Workshop on Software Architecture (ISAW'98)*, pages 69–72, November 1-2, 1998, Orlando, FL (USA). ACM.

[Holt, 1996] Holt, R. C. (1996). Binary relational algebra applied to software architectural. CSRI Technical Report 345, Computer Systems Research Institute, University of Toronto.

[Holt and Mancoridis, 1995] Holt, R. C. and Mancoridis, S. (1995). Using tube graphs to model architectural designs of software systems. Technical report, Department of Computer Science. University of Toronto (Canada).

[Huhn et al., 1995] Huhn, M., Wehrheim, H., and Denker, G. (1995). Action refinement - an application of process theory on object-oriented specification. Technical report, Hildesheimer Informatik-Berichte 40/95, Universität Hildesheim, Institut für Informatik, Postfach 101363, D-31113 Hildesheim.

[Huhn et al., 1996] Huhn, M., Wehrheim, H., and Denker, G. (1996). Action refinement in system specification: Comparing a process algebraic and an object-oriented approach. In Herzog, U. and Hermanns, H., editors, *Formale Beschreibungstechniken Für Verteilte Systeme*, pages 77–88.

[IEEE, 1994] IEEE (1994). *IEEE Software Engineering Standard Collection. 1994 Edition.* IEEE Computer Society Press.

[IEEE, 1995] IEEE (1995). IEEE standard for information technology - software reuse - data model for reuse library interoperability data model (BIDM). IEEE Std 1420.1, IEEE Computer Organization.

[Jacobson et al., 1999] Jacobson, I., Booch, G., and Rumbaugh, J. (1999). *The Unified Software Development Process.* Object Technology Series. Addison-Wesley.

[Jacobson et al., 1992] Jacobson, I., Christerson, M., Jonsson, P., and Övergaard, G. (1992). *Object-Oriented Software Engineering: A Use Case Driven Approach.* Addison-Wesley. (Revised 4th printing, 1993).

[Jacobson et al., 1994] Jacobson, I., Ericsson, M., and Jacobson, A. (1994). *The Object Advantage - Business Process Reengineering with Object Technology.* Addison-Wesley, Menlo Park, CA.

[Jacobson et al., 1997] Jacobson, I., Griss, M., and Jonsson, P. (1997). *Software Reuse. Architecture, Process and Organization for Bussiness Success.* ACM Press. Addison-Wesley.

[Jazayeri, 1995] Jazayeri (1995). Component programming - a fresh look at software components. In *5th European Software Engineering Conference (September 25-28 - Sitges, Spain).*

[Johannsson et al., 1996] Johannsson, P., Boman, M., Bubenko, J. J., and Wangler, B. (1996). *Conceptual Modelling.* Prentice-Hall.

[Johnson et al., 1993] Johnson, J. A., Nardi, B. A., Zarmer, C. L., and Miller, J. R. (1993). Ace: Building interactive graphical applications. *Communications of the ACM*, 36(4):40–55.

[Johnson, 1995] Johnson, R. E. (1995). Why doesn't the reuse community talk about reusable software? In *Proceedings of the 7th Workshop on Institutionalizing Software Reuse (WISR-7)*, Andersen Consulting's Center for Professional Education, St. Charles, Illinois, 28-30 August 1995. ftp://gandalf.umcs.maine.edu/pub/WISR/wisr7/proceedings/txt/johnson.txt.

[Johnson and Foote, 1988] Johnson, R. E. and Foote, B. (1988). Designing reusable classes. *Journal of Object-Oriented Programming*, 1(2):22–35.

[Johnson and Russo, 1991] Johnson, R. E. and Russo, V. F. (1991). Reusing object-oriented design. Technical Report UIUCDCS 91-1696, University of Illinois.

[Jones, 1984] Jones, T. C. (1984). Reusability in programming: A survey of the stateof the art. *Transactions on Software Engineering*, SE-10(5):488–494.

[Kang et al., 1992] Kang, K. C., Cohen, S., Holibaugh, R., Perry, J., and Peterson, A. S. (1992). A resuse-based software development methodology. Special Report CMU/SEI-92-SR-4, Software Engineering Institute. Carnegie Mellon University, Pittsburgh, Pennsylvania 15213.

[Kang et al., 1990] Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., and Peterson, A. S. (1990). Feature-Oriented Domain Analysis (FODA). Feasibility study. Technical Report CMU/SEI-90-TR21 (ESD-90-TR-222), Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, Pennsylvania 15213.

[Kara, 1997] Kara, D. (1997). The repository's role in component development. http://cool.sterling.com/cbd/kara.htm.

[Karlsson, 1995] Karlsson, E.-A., editor (1995). *Software Reuse. A Holistic Approach*. Wiley Series in Software Based Systems. John Wiley and Sons Ltd.

[Katz, 1990] Katz, R. (1990). Toward a unified framework for version modelling in engineering databases. *ACM Computing Surveys*, 22(4):375–408.

[Katz et al., 1994] Katz, S., Dabrowski, C., Miles, K., and Law, M. (1994). Glossary of software reuse terms. NIST Special Publication 500-222, National Institute of Standards and Technology, Gaithersburg, MD. http://sw-eng.falls-church.va.us/ReuseIC/pubs/reference/terminology.htm.

[Katz et al., 1989] Katz, S., Richter, C. A., and The, K.-S. (1989). PARIS: A system for reusing partially interpreted schemas. In [Biggerstaff and Perlis, 1989].

[Kleine, 1986] Kleine, K. (1986). Catalogue of data types. Technical Report TR 05.371, IBM Laboratory, Böblingen.

[Knuth, 1973a] Knuth, D. E. (1973a). *The Art of Computing Programming, Vol. 1; Fundamental Algorithms*. Addison-Wesley, 2nd edition.

[Knuth, 1973b] Knuth, D. E. (1973b). *The Art of Computing Programming, Vol. 3; Sorting and Searching*. Addison-Wesley.

[Kozaczynski and Booch, 1998] Kozaczynski, W. and Booch, G. (1998). Component-based software engineering. *IEEE Software*, 15(5):34–36.

[Kruchten et al., 1984] Kruchten, P., Schonberg, E., and Schwartz, J. T. (1984). Software prototyping using the SETL programming language. *IEEE Software*, 1(4):66–75.

[Krueger, 1992] Krueger, C. W. (1992). Software reuse. *ACM Computing Surveys*, 24(2):131–183.

[Le Métayer, 1996] Le Métayer, D. (1996). Software architecture styles as graph grammars. In *Proceedings of the Fourth ACM SIGSOFT Symposium on Foundations of Software Engineering (SIGSOFT '96)*, pages 15–23, October 16-18, 1996, San Francisco, CA (USA). ACM, ACM Press.

[Le Métayer, 1998] Le Métayer, D. (1998). Describing software architecture styles using graph grammars. *IEEE Transactions on Software Engineering*, 24(7):521–533.

[Lee and Stepanov, 1995] Lee, M. and Stepanov, A. A. (1995). *The Standard Template Library*. HP Labaratories, 1501 Page Mill Road, Palo Alto, CA 94304 (USA).

[Letelier et al., 1998] Letelier, P., Sánchez, P., Pastor, O., and Ramos, I. (1998). *OASIS Versión 3: Un Enfoque Formal Para el Modelado Conceptual Orientado a Objeto*. Servicio de Publicaciones UPV, Universidad Politécnica de Valencia. Valencia (Spain). SPUPV-98.4011.

[Lindqvist and Jonsson, 1998] Lindqvist, U. and Jonsson, E. (1998). A map of security risks associated with using COTS. *Computer*, 31(6):60–66.

[Liskov, 1987] Liskov, B. (1987). Data abstraction and hierarchy. In *Proceedings of OOPSLA'87*, pages 17–35. ACM Press. October 4-8. Addendum to the Proceedings.

[Lyon, 1981] Lyon, M. J. (1981). Salvaging your software asset (tools-based maintenance). In *Nat'l Computer Conference*, pages 337–341. AFIPS Press.

[Madany et al., 1991] Madany, P. W., Islam, N., Kougiouris, P., and Campbell, R. H. (1991). Reification and reflection in C++: An operating systems perspective. Technical report, Department of Computer Science, University of Illinois, Urbana-Champaign, 1304 W. Springfield Avenue, Urbana, IL 61801, USA.

[Mannion et al., 1999] Mannion, M., Kaindl, H., and Wheadon, J. (1999). Reusing single system requirements from application family requirements. In *Proceedings of the 1999 International Conference on Software Engineering (ICSE' 99)*, pages 453–462, May 16-22, 1999. Los Angeles, CA (USA). ACM.

[Manso et al., 1998] Manso, M. E., García, F. J., Rodríguez, J. J., and Laguna, M. Á. (1998). Modelo de cualificación de assets del repositorio GIRO. In [Moros and Sáez, 1998], pages 109–114.

[Manso et al., 1999a] Manso, M. E., García, F. J., Romay, M. P., and Marqués, J. M. (1999a). Modelo de cualificación y auditorías de elementos reutilizables de un repositorio. In Botella, P., Hernández, J., and Saltor, F., editors, *Actas de Las IV Jornadas de Ingeniería Del Software Y Bases de Datos (JISDB'99)*, pages 355–366, 24-26 Noviembre de 1999 - Cáceres (España).

[Manso et al., 1999b] Manso, M. E., Romay, M. P., and García, F. J. (1999b). Repository audit. In [García and Marqués, 1999], pages 11–15.

[Marqués, 1995] Marqués, J. M. (1995). *Jerarquías de Herencia en el Diseño de Software Orientado al Objeto*. PhD thesis, Universidad de Valladolid.

[Marqués, 1996] Marqués, J. M. (1996). Reutilización y diseño orientado al objeto: Informe de resultados y propuestas de investigación. In Toro, M., editor, *Actas de Las I Jornadas de Ingeniería Del Software*, pages 179–186, Sevilla (España), 14-15 de Noviembre de 1996. Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla.

[Marqués, 1998] Marqués, J. M. (1998). Soporte operativo para la reutilización del software: Repositorios y clasificación. In [Barros and Domínguez, 1998]. Ourense, del 6 al 10 de Julio de 1998.

[Marqués et al., 1994] Marqués, J. M., Cuesta, R., and de la Fuente, P. (1994). BLC++: A logical browser for C++. In Guasch and Huber, editors, *Proceedings of the Conference on Modelling and Simulation*, pages 362–366.

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

[Martínez and Maudes, 1997] Martínez, B. and Maudes, M. (1997). Desarrollo de componentes software reutilizables para el dominio del tratamiento digital de imágenes. Proyecto de 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.

[Matsumoto, 1989] Matsumoto, Y. (1989). Some experiences in promoting reusable software: Presentation in higher abstract levels. In Biggerstaff, T. J. and Perlis, A. J., editors, *Software Reusability. Applications and Experience*, volume II of *Frontier Series*, pages 157–185, New York. ACM Press. Addison Wesley.

[Maymir-Ducharme, 1997] Maymir-Ducharme, F. A. (1997). The product line business model. In *Proceedings of 8th Annual Workshop on Software Reuse WISR-8*, March 23-26, 1997. The Ohio State University, Columbus, Ohio (USA).

[McClure, 1996] McClure, C. (1996). Experiences from the OO playing field. *The Object World Insider*, 2(4):1–3.

[McClure, 1997] McClure, C. (1997). *Software Reuse Techniques: Adding Reuse to the System Development Process*. Prentice-Hall.

[McIlroy, 1976] McIlroy, M. D. (1976). Mass-produced software components. In [Buxton et al., 1976], pages 88–98.

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

[Meyer, 1985] Meyer, B. (1985). On formalism in specifications. *IEEE Software*, 2(1):6–26.

[Meyer, 1988] Meyer, B. (1988). *Object-Oriented Software Construction*. Prentice-Hall.

[Meyer, 1991] Meyer, B. (1991). *Eiffel: The Language*. Object-Oriented Series. Prentice-Hall. (Revised 2nd printing, 1992).

[Meyer, 1994] Meyer, B. (1994). *Reusable Software. The Base Object-Oriented Component Libraries*. The Object-Oriented Series. Prentice Hall.

[Meyer, 1997] Meyer, B. (1997). *Object Oriented Software Construction*. Prentice Hall, 2nd edition.

[Microsoft, 1997] Microsoft (1997). *Microsoft Repository Documentation*. Microsoft Corporation.

[Mili et al., 1995] Mili, H., Mili, F., and Mili, A. (1995). Reusing software: Issues and research directions. *IEEE Transactions on Software Engineering*, 21(6):528–562.

[Mili et al., 1997] Mili, R., Mili, A., and Mittermeir, R. T. (1997). Storing and retrieving software components: A refinement based system. *IEEE Transactions on Software Engineering*, 23(7):445–460.

[Moros and Sáez, 1998] Moros, B. and Sáez, J., editors (1998). *Actas de Las III Jornadas de Trabajo MENHIR*, 13 y 14 de Noviembre de 1998, Murcia (Spain). Organizado por el Grupo de Investigación de Ingeniería del Software de la Universidad de Murcia.

[Musser and Stepanov, 1989] Musser, D. R. and Stepanov, A. A. (1989). *The Ada Generic Library: Linear List Processing Packages*. Compass Series. Springer-Verlag.

[NATO, 1992a] NATO (1992a). NATO standard for management of a reusable software component library. Volume 2 (of 3 Documents). NATO Communications and Information Systems Agency (NACISA).

[NATO, 1992b] NATO (1992b). NATO standard for the development of reusable software components. Volume 1 (of 3 Documents). NATO Communications and Information Systems Agency (NACISA).

[Neighbors, 1984] Neighbors, J. M. (1984). The Draco approach to constructing software from reusable components. *IEEE Transactions on Software Engineering*, SE-10(5):564–574.

[Neighbors, 1991] Neighbors, J. M. (1991). Draco: A method for engineering reusable software systems. In [Prieto-Díaz and Arango, 1991], pages 34–52.

[NHSE, 1997a] NHSE (1997a). The internal data format of RIB.

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

[Oberndorf, 1998] Oberndorf, P. (1998). COTS and Open Systems. Sei monographs on use of commercial software in government systems, Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213.

[ObjectSpace, 1999] ObjectSpace (1999). JGL - generic collection library. http://www.objectspace.com/products/jglOverview.htm. [Consulta 17/9/99].

[OMG, 1998] OMG (1998). OMG unified modeling language specification. Version 1.2. OMG Document ad/98-12-02, Object Management Group Inc., 492 Old Connecticut Path, Framingham, MA 01701. ftp://ftp.omg.org/pub/docs/ad/98-12-02.pdf.

[OMG, 1999] OMG (1999). OMG unified modeling language specification. Version 1.3. Technical report, Object Management Group Inc. http://uml.shl.com:80/docs/UML1.3/99-06-08.pdf.

[Parnas, 1972] Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058.

[Parnas et al., 1989] Parnas, D. L., Clements, P. C., and Weiss, D. M. (1989). Enhancing reusability with information hiding. In [Biggerstaff and Perlis, 1989], pages 141–157.

[Partsch and Steinbruggen, 1983] Partsch, H. and Steinbruggen, R. (1983). Program transformation systems. *ACM Computing Surveys*, 15(3):199–236.

[Pastor and Ramos, 1995] Pastor, O. and Ramos, I. (1995). *OASIS Version 2 (2.2): A Class-Definition Language to Model Information Systems Using an Object-Oriented Approach*. Servicio de Publicaciones UPV, Universidad Politécnica de Valencia. Valencia (Spain). SPUPV-95.788.

[Peterson, 1991] Peterson, S. A. (1991). Coming to terms with software reuse terminology: A model-based approach. *ACM Software Engineering Notes*, 16(2):45–51.

[Petracca and Bock, 1994] Petracca, A. and Bock, T. (1994). Command center library model document release 4.0 comprehensive approach to reusable defense software (CARDS). Informal Technical Report for the Software Technology for Adaptable, Reliable Systems (STARS) STARS-VC-B015/002/00, STARS.

[Peuker, 1997] Peuker, T. (1997). An object-oriented architecture for real-time transmission of multimedia data streams. Technical report, Institut für Mathematische Maschinen und Datenvararbeitung (Informatik) IV. Universität Erlangen-Nürnberg.

[Plaza, 1997] Plaza, E. (1997). Noos language. http://www.iiia.csic.es/ ẽnric/noos/Overview.

[Poulin, 1997] Poulin, J. (1997). Software architectures, product lines, and DSSAs: Chossing the appropiate level of abstraction. In *Proceedings of 8th Annual Workshop on Software Reuse WISR-8*, March 23-26, 1997. The Ohio State University, Columbus, Ohio (USA).

[Poulin, 1999] Poulin, J. S. (1999). Reuse: Been there, done that. *Communications of the ACM*, 42(5):98–100.

[Pressman, 1997] Pressman, R. S. (1997). *Software Engineering: A Practitioner's Approach*. McGraw-Hill, 4th edition.

[Prieto-Díaz, 1989] Prieto-Díaz, R. (1989). Classification of reusable modules. In [Biggerstaff and Perlis, 1989], pages 99–123.

[Prieto-Díaz, 1991] Prieto-Díaz, R. (1991). Implementing faceted classification for software reuse. *Communications of the ACM*, 34(5):89–97.

[Prieto-Díaz, 1994] Prieto-Díaz, R. (1994). The disappearance of software reuse. In Frakes, W. B., editor, *Proceedings of the Third International Conference on Software Reusability ICSR-3*, page 225, Rio de Janerio, Brazil, 1-4 November 1994. IEEE Press.

[Prieto-Díaz and Arango, 1991] Prieto-Díaz, R. and Arango, G., editors (1991). *Domain Analysis and Software Systems Modeling*. IEEE Computer Society Press, Los Alamitos, California.

[Prieto-Díaz and Freeman, 1987a] Prieto-Díaz, R. and Freeman, P. (1987a). Classifying software for reusability. *IEEE Software*, 4(1):6–16.

[Prieto-Díaz and Freeman, 1987b] Prieto-Díaz, R. and Freeman, P. (1987b). Classifying software for reusability. *IEEE Software*, 4(1):6–16.

[Ragan and Reifer, 1998] Ragan, T. and Reifer, D. J. (1998). Adding product lines, architectures, and software reuse to the software acquisition capability maturity model. *Crosstalk*, 11(5):14–18.

[Ramos et al., 1995] Ramos, I., Pelechano, V., Penadés, M. C., Bonet, B., Canós, J. H., and Pastor, O. (1995). Análisis y diseño orientado a objetos de un entorno de prototipación automática basado en OASIS. Technical Report II-DSIC-9/95, Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia, Valencia (Spain).

[Ran, 1999] Ran, A. (1999). Software isn't built from lego blocks. In *Proceedings of the Fifth Symposium on Software Reusability (SSR'99)*, pages 164–169, May 21-23, 1999, Los Angeles, CA (USA). ACM, Inc.

[Reenskaug, 1996] Reenskaug, T. (1996). *Working with Objects. The OOram Software Engineering Method.* Manning/Prentice Hall.

[Rine and Nada, 1998] Rine, D. C. and Nada, N. (1998). Software reuse manufacturing reference model: Development and validation. http://www.gmu.edu/depts/survey/rrmpaper1.htm.

[Riva, 1998a] Riva, M. (1998a). SURF. achieving quality through software reuse: A process improvement experiment in IBM Italia. In *Proceedings of the 2nd European Reuse Workshop (ERW'98)*, pages 287–292, November, 4-6, 1998. Madrid (Spain). European Software Institute (ESI).

[Riva, 1998b] Riva, M. (1998b). SURF. Software re-Use: a process impRovement experiment at an IBM Italia Facility. Process Improvement Experiment Final Report ESSI - PIE Project 23752, European Systems & Software Initiative (ESSI). Version 1.0.

[Rivas et al., 1997] Rivas, E., DeSilva, D., McDaniel, T., and Atkinson, C. (1997). Object analysis and design facility. Response to OMG/OA&D RFP-1, Platinum Technology, Inc. Version 1.0.

[Rogerson, 1997] Rogerson, D. (1997). *Inside COM.* Microsoft Press.

[Romay et al., 1998] Romay, M. D. P., García, F. J., Crespo, Y., and Laguna, M. Á. (1998). Una experiecia práctica de reutilización: Puesta en marcha del repositorio GIRO. In [Moros and Sáez, 1998], pages 103–108.

[Rumbaugh et al., 1991] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W. (1991). *Object-Oriented Modeling and Design.* Prentice-Hall.

[Rumbaugh et al., 1999] Rumbaugh, J., Jacobson, I., and Booch, G. (1999). *The Unified Modeling Language Reference Manual.* Object Technology Series. Addison-Wesley.

[Sáez et al., 1999] Sáez, J., Fernández, J. L., Toval, A., Manso, M. E., Laguna, M. Á., and García, F. J. (1999). A reuse for formally verified UML diagrams. In [García and Marqués, 1999], pages 16–21.

[Sánchez et al., 1998] Sánchez, P., Letelier, P., Ramos, I., and Pastor, O. (1998). OASIS 3.0: Un enfoque formal para el modelado conceptual orientado a objeto. In [Moros and Sáez, 1998], pages 63–74.

[Sant'Anna et al., 1998] Sant'Anna, M., Do Prado Leite, J. C. S., and Do Prado, A. F. (1998). A generative approach to componentware. In *Proceedings of the 1998 Workshop on Component Based Software Engineering (CBSE'98)*, Kyoto (Japan).

[Schappert et al., 1995] Schappert, A., Sommerland, P., and Pree, W. (1995). Automated support for development with frameworks. In Samadzadeh, M. and Zand, M., editors, *Proceedings of the Symposium on Software Reusability (SSR'95) in the Special Issue of Software Engineering Notes (August, 1995)*, pages 123–127, April 28-30, 1995. Seattle, Washington. ACM, ACM Press.

[SEI, 1991] SEI (1991). Requirement engineering and analysis. Technical Report CMU/SEI-91-TR-30, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213.

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

[SER Consortium, 1996] SER Consortium (1996). Solutions for software evolution and reuse. SER Esprit Project 9809.

[Silva, 1998] Silva, A. (1998). Across Version/Variant requirement traceability in avionics software development and testing. In *Proceedings of the 2nd European Reuse Workshop (ERW'98)*, pages 179–198, November, 4-6, 1998. Madrid (Spain). European Software Institute (ESI).

[Simos and Anthony, 1998] Simos, M. and Anthony, J. (1998). Weaving the model web: A multi-modeling approach to concepts and features in domain engineering. In *Proceedings of the Fifth International Conference on Software Reuse (ICSR-5)*, pages 94–102, 2-5 June, 1998. Victoria, B. C., Canada. IEEE Computer Society, IEEE Press.

[Simos et al., 1996] Simos, M., Creps, D., Klingler, C., Levine, L., and Allemang, D. (1996). Organization domain modeling (ODM) guidebook - version 2.0. Technical Report STARS-VC-A025/001/00, Lockheed Martin Tactical Defense Systems, 9255 Wellington Road Manassas, VA 22110-4121.

[Solderitsch, 1992] Solderitsch, J. (1992). Making the case for interoperating reuse libraries. In *Proceedings of 5th Workshop on Institutionalizing Software Reuse (WISR5)*, Palo Alto, California.

[Solderitsch et al., 1992a] Solderitsch, J., Bradley, E. A., and Schreyer, T. M. (1992a). The reusability library framework - levering software reuse. In *Proceedings of STARS'92*.

[Solderitsch et al., 1992b] Solderitsch, J., Thalhamer, J., and Creps, D. (1992b). Asset library open architecture framework - sharing reusable assets. In *Proceedings of DARPA Software Technology Conference '92*, pages 242–249, Los Angeles, California - April 1992.

[Sonnemann, 1995] Sonnemann, R. (1995). *Exploratory Study of Software Reuse Factors*. PhD thesis, George Mason University, Fairfax, Virginia, Spring.

[Sorumgard et al., 1993] Sorumgard, S., Stokke, F., and Sindre, G. (1993). Experiences from the application of a faceted classification scheme. In Guerrieri, E., editor, *Proceedings of International Workshop on Software Reuse (ISRW-2)*, March 24-26, 1993. Lucca (Italy).

[Sparks et al., 1996] Sparks, S., Benner, K., and Faris, C. (1996). Managing object-oriented framework reuse. *Computer*, 29(9):52–61.

[STARS, 1993] STARS (1993). STARS Conceptual Framework for Reuse Processes (CFRP). Technical Report STARS-VC-A018/001/00, STARS. Version 3.0 Vols I & II.

[Stein, 1987] Stein, L. A. (1987). Delegation is inheritance. In *Proceedings of OOPSLA'87*, pages 138–146. ACM Press. October, 4-8.

[Stroustrup, 1997] Stroustrup, B. (1997). *The C++ Programming Language*. Addison-Wesley, Reading, Massachusetts, third edition.

[Szyperski, 1995] Szyperski, C. (1995). Component-oriented programming a refined variation on object-oriented programming. *The Oberon Tribune*, 1(2).

[Tracz, 1988a] Tracz, W., editor (1988a). *IEEE Tutorial: Software Reuse Emerging Technology*. The Computer Society Press, 10662 Los Vaqueros Circle P.O. Box 3014. Los Alamitos, CA 90720-1264.

[Tracz, 1988b] Tracz, W. (1988b). Software reuse: Motivators and inhibitors. In [Tracz, 1988a], pages 62–67.

[Tracz, 1988c] Tracz, W. (1988c). Software reuse myths. In [Tracz, 1988a], pages 18–22.

[Tracz, 1990] Tracz, W. (1990). Where does reuse start? In *Proceedings of the Realities of Reuse Workshop*, Syracuse University CASE Center.

[Tracz, 1994] Tracz, W. (1994). Software reuse myths revisited. In *Proceedings of the 16th International Conference on Software Engineering (ICSE'94)*, pages 271–272, May 16-21, 1994, Sorrento (Italy). IEEE Computer Society.

[Tracz et al., 1993] Tracz, W., Coglianese, L., and Young, P. (1993). Domain-specific SW architecture engineering. *Software Engineering Notes*, 18(2):40–49.

[Tracz and Edwards, 1989] Tracz, W. J. and Edwards, S. H. (1989). Implementation working group report. In *Proceedings of the Reuse in Practice Workshop*, Pittsburgh, PA.

[Trump, 1997] Trump, D. (1997). Using the WWW and the internet to support corporate reuse. In *Proceedings of the 8th Annual Workshop on Software Reuse WISR-8*, March 23-26, 1997. The Ohio State University, Columbus, Ohio (USA).

[Uhl and Schmid, 1990] Uhl, J. and Schmid, H. A. (1990). *A Systematic Catalogue of Reusable Abstract Data Types*. Lecture Notes in Computer Science. Springer-Verlag. Edited by G. Goos and J. Hartmanis.

[Veryard, 1997] Veryard, R. (1997). Component-based development. http://www.veryard.com.

[Viasoft Inc., 1997] Viasoft Inc. (1997). The Rochade repository environment. The foundation for the information asset warehouse. White paper, Viasoft Incorporation. http://www.viasoft.com/rochade/whitepaper/.

[Villa, 1997] Villa, E. (1997). 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 Redondo y José Manuel Marqués Corral.

[Voas, 1998] Voas, J. M. (1998). The challenges of using COTS software in component-based development. *Computer*, 31(6):44–45.

[Wallnau, 1992] Wallnau, K. C. (1992). Towards an extended view of reuse libraries. In *Proceedings of 5th Workshop on Institutionalizing Software Reuse (WISR-5)*, Palo Alto, California (USA).

[Warmer and Kleppe, 1999] Warmer, J. B. and Kleppe, A. G. (1999). *The Object Constraint Language: Precise Modeling with Uml*. Object Technology Series. Addison-Wesley.

[Wartik and Prieto-Díaz, 1992] Wartik, S. and Prieto-Díaz, R. (1992). Criteria for comparing domain analysis approaches. *International Journal of Software Engineering and Knowledge Engineering*.

[Wasmund, 1994] Wasmund, M. (1994). Reuse facts and myths. In *Proceedings of the 16th International Conference on Software Engineering (ICSE'94)*, page 273, May 16-21, 1994, Sorrento (Italy). IEEE Computer Society.

[Web, 1997] Web, P. C. (1997). Web dictionary of cybernetics and systems. http://pespmc1.vub.ac.be/ASC.

[Wentzel, 1994] Wentzel, K. D. (1994). Software reuse - facts and myths. In *Proceedings of the 16th International Conference on Software Engineering (ICSE'94)*, pages 267–268, May 16-21, 1994, Sorrento (Italy). IEEE Computer Society.

[Withey, 1996] Withey, J. (1996). Investment analysis of software assets for product lines. Technical Report CMU/SEI-96-TR-010 (ESC-TR-96-010), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA).

[Wolf, 1997] Wolf, G. (1997). Steve jobs: The next insanely great thing. The Wired Interview. *Wired*, 3(2).

[Yourdon, 1996] Yourdon, E. (1996). Lipstick on a pig or real silver bullet? *American Programmer*, 9(11):25–29.