MECANO: UNA PROPUESTA DE COMPONENTE SOFTWARE REUTILIZABLE[*]

Francisco José García Peñalvo
Departamento de Ingeniería Electromecánica y Civil
Área de Lenguajes y Sistemas Informáticos
Escuela Universitaria Politécnica, Universidad de Burgos
Av. General Vigón S/N, 09006 Burgos (España)
☎ +34 47 258989
e-mail: fgarcia@ubu.es

José Manuel Marqués Corral
Departamento de Informática
Escuela Universitaria Politécnica, Universidad de Valladolid
C/ Francisco Mendizabal, 1, 47014 Valladolid (España)
☎ +34 83 423502
e-mail: jmmc@dcs.eup.uva.es

Jesús Manuel Maudes Raedo
Departamento de Ingeniería Electromecánica y Civil
Área de Lenguajes y Sistemas Informáticos
Escuela Universitaria Politécnica, Universidad de Burgos
Av. General Vigón S/N, 09006 Burgos (España)
☎ +34 47 258989
e-mail: jmaudes@ubu.es

RESUMEN

La reutilización sistemática del software es un camino hacia la mejora de la calidad del software y el aumento de productividad a la hora de construirlo. Para llevar a cabo esta reutilización se deben definir claramente las estructuras de reutilización, esto es, los componentes software reutilizables de diferente nivel de abstracción (*assets*) y sus interdependencias, generando un metamodelo de elemento reutilizable complejo que pueda ser gestionado por un repositorio. En el presente trabajo se va presentar una propuesta de estructura compleja de reutilización, la cual será referenciada con el nombre de mecano.

PALABRAS CLAVE

Investigación, reutilización de software, componentes software, relaciones semánticas, metamodelo, repositorio

AGRADECIMIENTOS

Este documento ha sido generado en el seno del grupo GIRO^2 (*Grupo de Investigación en Reutilización y Orientación al Objeto*) compuesto por miembros de la E.U.P de la Universidad de Valladolid y de la E.U.P de la Universidad de Burgos. Desde aquí queremos agradecer la inestimable colaboración y las correcciones sugeridas por el resto de los miembros de este grupo.

---

[*] Trabajo publicado en las actas de las II Jornadas de Ingeniería del Software, Donostia-San Sebastián, 3-5 de Septiembre de 1997, pp:232-244.

1. INTRODUCCIÓN

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. Esto motiva que los esfuerzos en investigación y desarrollo se centren en aquellas técnicas y herramientas que mejoren el desarrollo eficiente de sistemas software.

La reutilización del software ha sido, y sigue siendo, uno de los principales temas de investigación en el campo de la Ingeniería del Software, citándose a menudo como una de las principales técnicas para incrementar la productividad de los desarrolladores de software. Así 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*". Sin embargo, este mismo autor reconoce que no se han conseguido grandes avances en la adopción sistemática de la reutilización en el proceso de construcción del software.

De un tiempo a esta parte se han producido grandes anuncios acerca de "*la solución definitiva*" a todos los problemas de la Ingeniería del Software. Por citar algunos ejemplos de las denominadas balas de plata de la ingeniería del software (Brooks 1987), se podrían mencionar, entre otros, los *lenguajes de cuarta generación*, la *tecnología CASE* y la *Orientación al Objeto*. Estas promociones en muchos casos lo único que encierran son intereses comerciales (*se han llegado a vender compiladores tradicionales como herramientas "lower CASE"*), y en ningún caso proporcionarán por sí solas la solución a la tan nombrada crisis del software.

El camino hacia el desarrollo de software de calidad con una mayor productividad pasa por la adopción conjunta de técnicas de reutilización sistemática, tecnología de Orientación al Objeto, metodologías de desarrollo soportadas por métodos formales, entornos RAD y herramientas CASE adecuadas.

Según estas ideas se está trabajando en el grupo de investigación (GIRO^2)[1] en aspectos relacionados con la reutilización (*browser lógico* (Marqués et al.1994), *herencia* (Marqués 1996), *evolución de esquemas* (Maudes et al. 1997)), con el fin de integrarlos dentro de los trabajos del grupo OASIS del DSIC de la *Universidad Politécnica de Valencia*, centrados desde hace algunos años en entornos de programación automática, contando ya con un lenguaje de especificaciones formales, OASIS (Pastor y Ramos 1995), una propuesta metodológica, OASIS Method (Toval et al. 1996), y un entorno general de producción automática de prototipos de software orientado al objeto (Ramos y Toval 1995), (Penadés et al. 1996).

2. ELEMENTOS SOFTWARE REUTILIZABLES

La reutilización del software no es un concepto nuevo, de hecho se lleva reutilizando software durante varias décadas. El término reutilización fue originalmente acuñado en 1968 por M.D. McIloroy (McIlroy 1976). Inicialmente el concepto de

[1] **Grupo de Investigación en Reutilización y Orientación al Objeto**, formado por personas pertenecientes a la *Escuela Universitaria Politécnica de la Universidad de Valladolid* y a la *Escuela Politécnica de la Universidad de Burgos.*

reutilización se concibe como la combinación de componentes de código almacenados en una biblioteca, limitando el proceso de reutilización a la reutilización del código, y llevada a cabo de una forma caracterizada por una total carencia de tintes metodológicos.

Actualmente el concepto de reutilización ha evolucionado hacia la idea de que todo el conocimiento y productos derivados de la producción de software son susceptibles de ser reutilizados en la construcción de nuevos sistemas (Freeman 1987a), surgiendo de esta forma el concepto de *asset* o de *componente software reutilizable* (Karlsson 1995). Así, se va a entender por *elemento reutilizable* cualquier producto software obtenido en el ciclo de vida del software, con independencia de su nivel de abstracción.

Según lo expresado anteriormente, la reutilización se puede definir como "*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 1987b) o "*utilizar elementos software existentes durante la construcción de un nuevo sistema software*" o "*utilización de conceptos y objetos existentes en un sistema o situación nueva, directamente o adaptándalos. Para ello, estos conceptos y objetos deberán encontrarse codificados en un nivel de abstracción establecido y deberán poder ser recuperados*" (Krueger 1992).

La extensión del concepto de producto software reutilizable, ha provocado la necesidad de clasificar los trabajos relacionados con la reutilización según el objeto y el método de reutilización usado. Diferentes clasificaciones se pueden encontrar en (Krueger 1992), (Mili et al. 1995), (Jones 1984), (Edwards et al. 1997). Sin embargo, la forma más habitual de encuadrar los trabajos de reutilización en la literatura es atendiendo a una clasificación muy simple según la fase del desarrollo y/o el nivel de abstracción en el que el conocimiento se produce o reutiliza. Este doble criterio establece la reutilización según tres niveles: *reutilización de código, reutilización de diseños* y *reutilización de especificaciones*.

Si algo se puede concluir del estudio de todas estas taxonomías es que todas ellas conducen hacia unos componentes reutilizables definidos sólo en un único nivel de abstracción o en una sola etapa del ciclo de vida del software.

La reutilización de código es la forma más común y extendida de reutilización. Se realiza durante la fase de implementación, pudiendo ser objeto de la reutilización el código fuente, el código objeto, bibliotecas estándar... El proceso de reutilización se realiza mediante editores, inclusión de ficheros, mecanismos de herencia en entornos de programación con orientación al objeto, llamadas a las rutinas de una biblioteca... La reutilización en este nivel está limitada por la dependencia del lenguaje, del sistema operativo, de la aplicación... La forma más tradicional de reutilización de código es la que se basa en bibliotecas de funciones, la aplicación llama a las funciones de la biblioteca a través de una capa de interfaz para conseguir los servicios que necesita, cuando los necesita (Sparks et al. 1996).

El aumento de la potencia de reutilización pasa por la elevación del nivel de abstracción. La abstracción constituye uno de los elementos fundamentales en la reutilización. David Parnas (Parnas et al. 1989) afirma que el desarrollo y catalogación de abstracciones incrementa la capacidad de reutilización del software desarrollado, ya que los desarrollos de una abstracción pueden ser reutilizados para cualquier modelo válido de la abstracción. La reutilización, por lo tanto, tiene que enfocarse hacia la reutilización de requisitos y diseños de alto nivel, de esta forma se elimina la necesidad

de volver a inventar arquitecturas y además se posibilita la reutilización de análisis y conjuntos de requisitos, que pueden ser adecuados para los nuevos sistemas a desarrollar.

En cuanto al nivel de reutilización de especificaciones, corresponde a la reutilización de las abstracciones de más alto nivel, aquellas que están relacionadas con el conocimiento del dominio. La incorporación de la reutilización sistemática desde la fase de especificación de requisitos, reduce el esfuerzo necesario para trasladar los conceptos iniciales del sistema a su forma final ejecutable. Para que este proceso pueda llevarse a cabo es necesario que la reutilización de requisitos esté asociada a la reutilización o generación automática o semiautomática de los elementos de diseño (*esquemas de aplicación*) e implementación correspondientes, de forma que permitan obtener un sistema software ejecutable y robusto.

Cuando se realizan desarrollos software que implican reutilización, ya sean desarrollos para reutilización o desarrollos con reutilización, se tiende a reutilizar componentes atómicos dentro de las categorías anteriormente descritas, los cuales se encuentran debidamente almacenados y clasificados dentro de algún repositorio[2]. Esto conduce a la conclusión de que ni el desarrollador para reutilización ni el que lo hace con reutilización tienen en mente la concepción de un componente reutilizable complejo que reúna assets de diferentes niveles de abstracción.

La idea que se persigue en este trabajo es precisamente la de ver al componente reutilizable como un subsistema que implicaría la agregación de varios componentes atómicos relacionados entre sí y clasificados en todos los niveles de abstracción y en todas las etapas del ciclo de vida, de forma que se daría opción a la reutilización de los subsistemas completos per se, además de seguir permitiendo la reutilización de los elementos atómicos cuando éstos sean necesarios.

3. INTRODUCCIÓN DEL CONCEPTO DE MECANO

Desde el punto de vista de conseguir entornos que permitan llevar a cabo proyectos con el mayor grado de automatización posible, pero apoyándose en la reutilización, sería de vital importancia contar con componentes reutilizables que cubran todo el ciclo de vida del software. Bajo estas perspectivas el grupo GIRO trabaja en la definición de una estructura de reutilización compleja, que recibe el nombre de *mecano*, y que además pueda ser implementable en un gestor de bases de datos orientadas al objeto.

Estas consideraciones sugieren la introducción de un componente reutilizable, cuya estructura incorpore elementos software de diferentes niveles de abstracción. Este componente software, el cual va a ser referenciado de aquí en adelante con el nombre de

[2] A la hora de llevar la reutilización a la práctica las tendencias actuales son establecer repositorios que permitan el trabajo conjunto de equipos de desarrollo localizados en diferentes lugares geográficos, utilizando las facilidades que ofrece Internet. La utilización de la flexibilidad y las prestaciones que ofrece la tecnología web puede aprovecharse para crear un entorno que de soporte al desarrollo con reutilización y para la reutilización de forma distribuida a través de Internet (Trump 1997). A modo de ejemplo cabe destacar herramientas como MOREplus o Microsoft Repository, así como el incremento del número de bibliotecas de assets disponibles en Internet, siendo ASSET, ELSA (Eichmann et al. 95), the RueseIC, y CARDS algunos de los más notables repositorios basados en tecnología web disponibles en Internet.

mecano, estará compuesto por elementos especificación, diseño, e implementación. Los mecanos se diseñarán lo suficientemente abstractos como para que puedan ser aplicados a una gran variedad de problemas, conteniendo la suficiente información asociada y relacionada como para que el usuario pueda adaptarlos al problema particular sin necesidad de comprender todos los detalles de bajo nivel.

La estructura del mecano estará representada por un grafo, donde los nodos corresponden a los diferentes componentes del mecano y los arcos a las relaciones existentes entre estos componentes. Los mecanos son sistemas complejos que están compuestos por varios componentes software de diferentes niveles de abstracción. Para la construcción y mantenimiento de nuevos sistemas basados en mecanos se debe tener muy claro las dependencias existentes entre los componentes que forman cada uno de los mecanos.

Para dar soporte a la reutilización, los componentes deben ser diseñados de forma que minimicen las dependencias. Pero cuando el acoplamiento es necesario, como sucede en el caso de los mecanos, las dependencias deben expresarse de forma clara y precisa (Edwards et al. 1997).

Uno de los factores claves del éxito o fracaso del proceso de construcción de software soportado por la reutilización es la existencia de un entorno que permita una clasificación y recuperación de componentes de forma rápida y efectiva. Esto supone que el entorno debe estar dotado de elementos conceptuales y herramientas que automaticen total o parcialmente este proceso.

Como primer paso se debe establecer el modelo de mecano que será soportado por el repositorio que almacenará los diferentes componentes de dichos mecanos, así como sus interdependencias.

Para definir este modelo se van a seguir una serie de pasos, que son: *identificación de los assets que pueden ser componentes de un mecano, identificación de las relaciones posibles entre los diferentes assets, creación de una lista con los elementos concretos que pueden formar los mecanos, creación del modelo de mecano.*

3.1 Tipos de assets

La forma más sencilla y natural de clasificar los assets es atendiendo a la fase del ciclo de vida donde fueron generados, así se tendrían tres categorías: *especificación, diseño* y *implementación*. En la categoría de especificación tendría cabida el modelo de especificación de requisitos, por ejemplo un modelo realizado mediante un lenguaje formal de especificación como puede ser OASIS. En la categoría de diseño entrarían todo tipo de especificaciones y elementos de diseño, por ejemplo un esquema de una base de datos o un pattern. La última categoría se correspondería con el modelo de implementación, por ejemplo clases C++, fuentes Java, componentes visuales...

Se puede añadir una segunda dimensión, ortogonal a la anterior, por la que se pueden clasificar los componentes en *abstractos* y *concretos*. Los componentes abstractos harían referencia a las especificaciones, centrándose en el comportamiento funcional. Los componentes concretos serían las implementaciones, es decir cómo se ofrecen los servicios.

Por último se podría añadir una tercera dimensión, ortogonal a las dos anteriores, que clasificase los componentes en *plantillas* e *instancias*.

Los tres ejes que representan las dimensiones de clasificación, como se puede apreciar en la Figura 1, forman un espacio tridimensional, en el que se encuentran los elementos reutilizables. Este espacio tridimensional no puede considerarse de forma discreta, ya que existiría una pérdida semántica al clasificar cualquier elemento simplemente como una 3-tupla compuesta de un elemento de cada dimensión. Por ejemplo, una biblioteca de clases si se considera como un todo, sería un componente reutilizable que vendría caracterizado por la triada <*implementación, concreto, instancia*>, pero si se intentase clasificar cada uno de los componentes de la biblioteca existirían elementos que se acomodarían en diferentes lugares de dicho espacio tridimensional (véase una clase abstracta en C++ <*implementación, abstracta,* instancia> y una clase parametrizada en C++ <*implementación, concreta, plantilla*>). Este problema sugiere la necesidad de elaborar criterios para la medida de la distancia continua entre los componentes en dicho espacio tridimensional, lo cual complementaría el proceso de búsqueda, no obstante dicho criterio queda fuera del ámbito del presente trabajo.

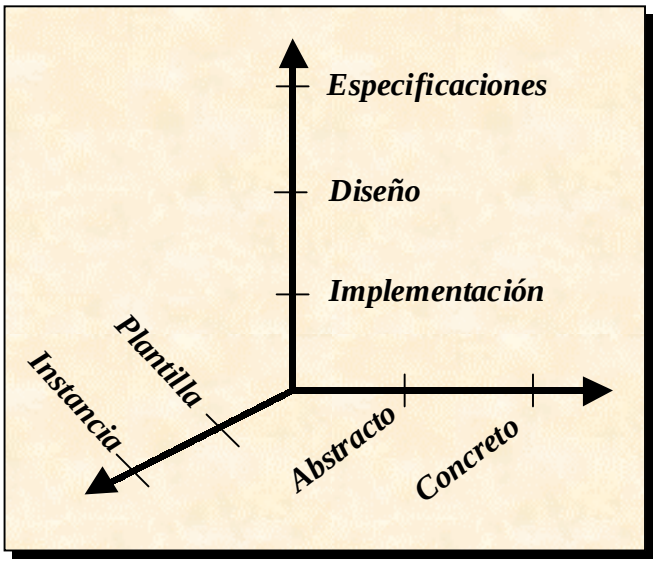

Figura 1. Dimensiones para la clasificación de los assets componentes de un mecano reutilizable.

3.2 Relaciones entre los componentes

Existen diferentes aproximaciones a la hora de modelar las dependencias y relaciones entre los assets, como ya se comentó anteriormente. Algunas de las más notables son las propuestas por ITHACA, por Randy Katz, por Graddy Booch, por Andrew Girow o por UML (Unified Modeling Language) (Rational 1997a).

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).

Graddy Booch mantiene que existen tres tipos básicos de relaciones: *especialización/generalización, parte de* y *asociación.* Estos tres tipos básicos dan lugar, gracias a la influencia de los lenguajes de programación, a otra serie de relaciones: *asociación, herencia, agregación, uso* e *instanciación* (Booch 1996).

Andrew Girow señala como relaciones principales: *generalización, agregación* y *asociación (*Girow 1996a), (Girow 1996b).

En UML se presentan cinco tipos fundamentales de relaciones: *dependencia, generalización, asociación, transición* y *enlace*. Siendo las dos más importantes la generalización y la asociación, definiendo la agregación como un tipo de asociación (Rational 1997b).

En el enfoque que aquí se ha dado, se distinguen tres tipos de relaciones: *relaciones semántico-estructurales generales* (que incluyen *ser atributo de, clasificación* y *generalización*), *relaciones semántico-estructurales especializadas* (que incluyen *agregación, correspondencia, similitud* y *especificidad*) y *asociaciones* (Constantopoulos et al. 92). Además, en el modelo que se propone las relaciones vienen caracterizadas a través de un conjunto de atributos y un conjunto de reglas ECA (*reglas evento-condición-acción*) (Paton et al. 1993), (Díaz y Paton 1994), (Díaz 1995), (Maudes et al. 1997). Esto proporciona un mecanismo de extensión de las relaciones a partir de las siguientes relaciones base:

- Especialización/Generalización. Es una relación de clasificación entre un elemento más general y un elemento más específico. El elemento más específico es completamente consistente con el elemento más general, conteniendo información adicional (Rational 1997c).
- Asociación. Describe un grupo de enlaces con estructura y semántica comunes (Rumbaugh et al. 1996). Debe implicar a dos o más elementos.
- Instanciación. Relación que implica la creación de instancias. Presenta el paso de un elemento genérico a un elemento concreto (Booch 1996).
- Dependencia. Indica una situación en la cual un cambio en un elemento destino puede requerir un cambio en el elemento fuente de la dependencia (Rational 1997d). La relación de equivalencia propuesta por Katz (Katz 1990) puede ser vista como una dependencia bidireccional. Las instancias de una relación de dependencia se pueden enriquecer con las funciones de paso del elemento fuente al destino de la dependencia, con lo que se puede mecanizar la sincronización de los cambios entre elementos dependientes (una muestra de esto es el Roundtrip Engineering (Yourdon 1997)).
- Uso. Relación semántica que supone un refinamiento de una asociación, por el que se establece qué abstracción es el cliente y qué abstracción es el servidor que proporciona ciertos servicios (Booch 1996).
- Agregación. Es una forma especial de asociación que especifica una relación todo/parte entre el componente agregado (todo) y sus partes componentes (parte) (Rational 1997c).
- Referencia a versión. Es la relación que mantiene un objeto de diseño con una versión[3] de un objeto de diseño. Esta relación puede ser estática o dinámica (Katz 1990).
- Composición. Relación semántica que describe una forma de agregación con un matiz fuerte de propiedad y de coincidencia de vida como partes de un todo. Las partes con una multiplicidad que no es fija pueden ser creadas después del objeto compuesto, pero una vez creadas ellas viven y mueren con él. Estas partes pueden ser también eliminadas antes de la muerte del objeto compuesto. La composición puede ser recursiva (Rational 1997c).

[3] Una versión de un objeto es una instantánea semánticamente significativa del objeto, tomada en un momento dado en el tiempo (Bertino 1993), (Katz 1990).

3.3 Una primera aproximación al modelo de mecano

El modelo del mecano que se muestra a continuación no es más que una propuesta inicial. El objetivo del presente modelo no es alcanzar la compleción, sino servir como elemento inicial de trabajo. Además la visión del modelo gráfico del mecano ayuda a la mejor comprensión de la estructura del mismo. El modelo que aparece en la Figura 2 utiliza la notación UML (Rational 1997c).

4. REPOSITORIO GIRO[2]

Los mecanos reutilizables se deben almacenar en un repositorio. Se entiende por repositorio una base de datos de información compartida sobre los elementos que se producen o se usan en un desarrollo software (Bernstein y Dayal 1994).

La problemática del modelo del mecano reutilizable es que ha de tener una estructura evolutiva que permita definir los elementos software en función de las tendencias del momento. Esto obliga a la creación de un metamodelo que permita extender y modificar el modelo.

Un metamodelo puede definirse como "*el modelo de información para la información que puede ser expresada durante el modelado*" (Metamodel 1997), o como "*un modelo que define el lenguaje para expresar un modelo*" (Rational 1997c). Por su parte se define modelo como "*la abstracción de un sistema semánticamente cerrada*" (Rational 1997c).

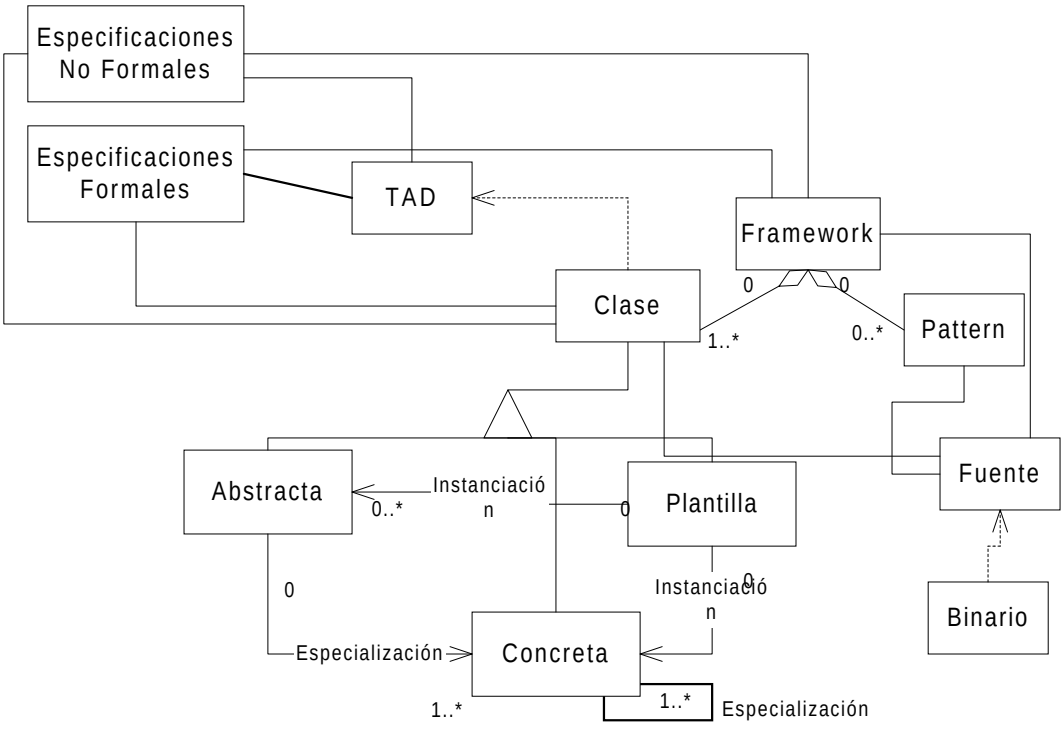

Figura 2. Propuesta de modelo de mecano.

Un metamodelo ha de permitir definir cualquier modelo en términos de los elementos del metamodelo, así pues dichos elementos no pueden evolucionar ni extenderse, pero a cambio se tiene la posibilidad de modificar y extender los elementos del modelo.

En términos de la notación de UML (Rational 1997d) se presenta el metamodelo que refleja la estructura de un mecano reutilizable, y a partir del cual se van a poder definir los modelos de esquema de repositorio específicos que se deseen, ver Figura 3.

Como paso previo a la definición y desarrollo del repositorio conviene tener en mente el concepto que se quiere modelar, esto es la estructura del mecano reutilizable, y en que términos se va a llevar a cabo: *los assets del repositorio contienen una serie de subsistemas de información que ayudan a su clasificación, y se relacionan entre sí para dar vida a los mecanos reutilizables.*

Se entiende por asset cualquier elemento software reutilizable, generado durante cualquier fase del ciclo de vida del desarrollo del software (*especificaciones, diseños, código, documentación, pruebas…*). Un asset genérico tiene una serie de atributos característicos y esenciales, y una serie de subsistemas que van a contribuir a complementar la información que de ellos se tiene, influyendo especialmente en temas de clasificación.

Los atributos que se consideran esenciales en un asset son:

- Identificador: Identificador único de un asset en el repositorio.
- Nombre: Nombre o título del asset.
- Versión: Identificación de versión.
- Fecha Creación: Fecha de creación.
- Fecha Modificación: Fecha de última modificación.

En lo que respecta a los subsistemas de información, se han identificado seis: descriptor, cualificación, administrativo, técnico, documentación y pruebas.

El subsistema descriptor tiene como objetivo implementar el mecanismo de clasificación (*facetas, información del dominio...*). Esta información será la que se utilice a la hora de seleccionar al asset para su posible recuperación.

El subsistema cualificación recogerá la información relativa a la calidad y la evaluación del asset, esto es, la serie de métricas que se crean necesarias y el nivel de evaluación puesto por el administrador del repositorio.

El subsistema administrativo se utiliza para almacenar toda la información relevante sobre el autor o autores, organización, permisos de accesos al asset...

El subsistema técnico puede utilizarse para recoger datos sobre la herramienta en que se generó el asset, el formato en que se encuentra...

El subsistema documentación recoge todos los documentos, guías... que se tengan sobre el asset.

El subsistema pruebas recoge los juegos de pruebas para el asset.

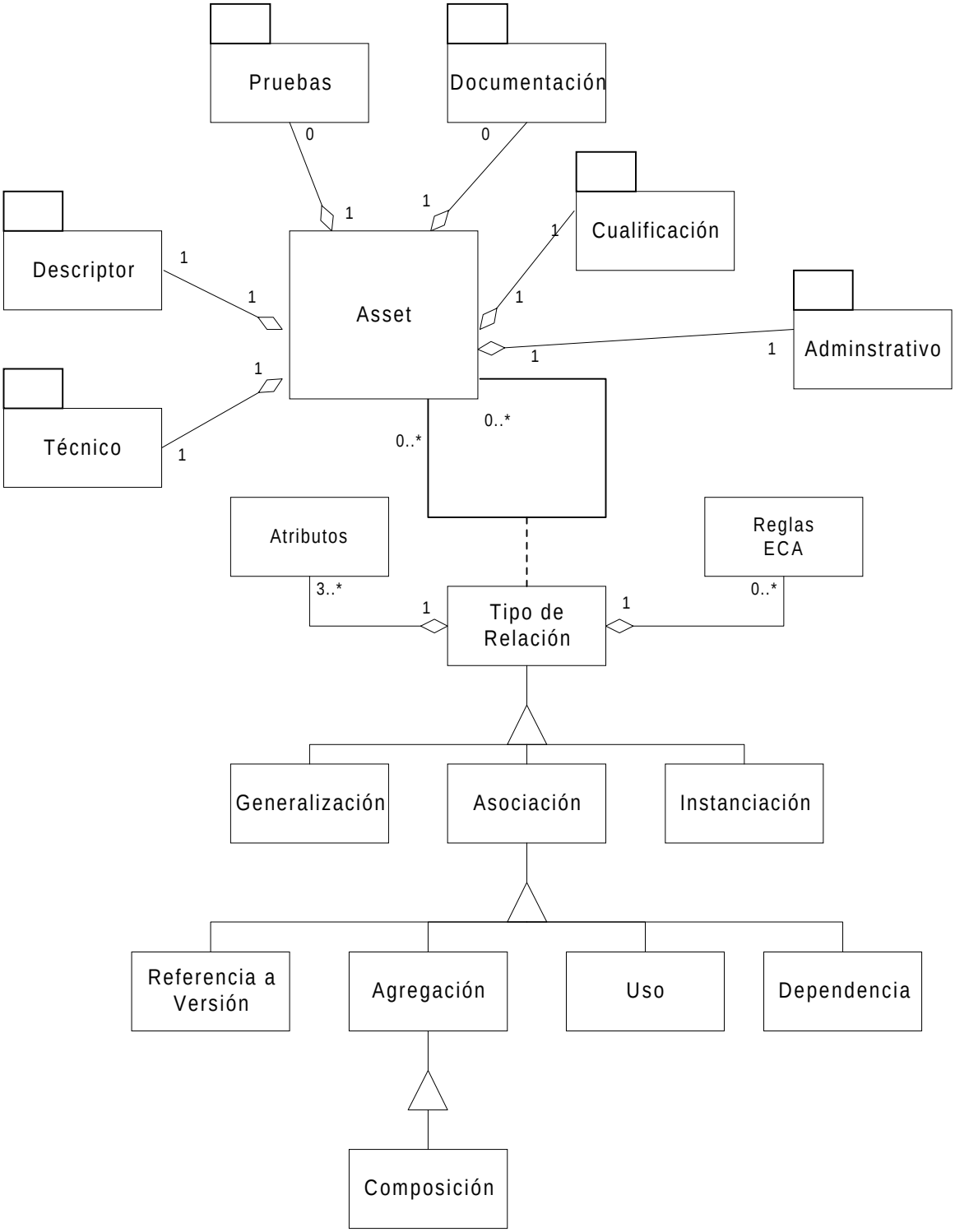

Figura 3. Metamodelo del esquema del repositorio.

5. CONCLUSIONES Y TRABAJO FUTURO

En este trabajo se han presentado una propuesta de definición de estructura del mecano reutilizable y de su soporte por un repositorio, y lo que es más importante se han sentado las bases para seguir trabajando en el proyecto de los mecanos reutilizables y seguir investigando dentro del apartado de la Ingeniería del Software dedicado a la Reutilización, pero con miras a un entorno de programación automática.

Los siguientes, e inminentes, pasos a seguir pasan por concretar los subsistemas del asset, especialmente el subsistema descriptor y el subsistema cualificación, así como establecer un modelo de repositorio cerrado que se ajuste al metamodelo aquí expuesto, y con el que se pueda empezar a realizar pruebas reales. Para este primer repositorio se utilizara EUROWARE. El diseño de la herramienta está pensada para la gestión de la reutilización de componentes por usuarios geográficamente distribuidos a través de Internet y clientes WWW, aunque en una segunda etapa se utilizará un gestor de BDOO como Postgres para su implementación práctica.

Otra actividad que se piensa abordar es la referente a la evolución y versionado de los mecanos. Como se ha venido indicando a lo largo de este documento, el concepto de mecano contiene todos los elementos correspondientes a un proceso de desarrollo. Estos procesos están sometidos a cambios constantes como consecuencia de cambios en el dominio o en la tecnología. Estos cambios van a provocar la necesidad de modificaciones en los contenidos y estructura del mecano. Por ejemplo, la definición de nuevos tipos de relaciones en el mecano implicará la integración del nuevo tipo de relación en el modelo. Si el repositorio GIRO sirviera de base al desarrollo de entornos de reutilización, que lo integraran en su arquitectura, se necesitaría que las instancias del repositorio fueran visibles y actualizables desde cualquier modelo del repositorio, con independencia del modelo al que realmente pertenezcan las instancias del repositorio, esto es, versionado total de esquemas según Roddick (Roddick 1995). Esta tarea estará centrada en el estudio de la evolución y versionado de los mecanos contemplados desde una perspectiva de esquema de Base de Datos Orientada al Objeto.

6. REFERENCIAS

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

Bertino, Elisa, Martino Lozarenzo. 1993. "*Object-Oriented Database Systems: Concepts and Architectures*". Addison-Wesley.

Booch, Grady. 1996. "*Análisis y Diseño Orientado a Objetos con Aplicaciones*". 2ª Ed. Addison-Wesley/Díaz de Santos.

Brooks, F. 1987. "*No Silver Bullet: Essence and Accident of Software Engineering*". IEEE Computer, Vol. 20, Nº4 (April): 12.

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

Díaz, Oscar, Paton, N.W. 1994. "*Extending ODBMSs Using Metaclasses*". IEEE Software, May 1994: 40-48.

Díaz, Oscar. 1995. "*The Operational Semantics of User-Defined Relatioships in Objetct Oriented Database Systems*". Data & Knowledge Engineering. Vol. 16: 223-240. North-Holland,

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

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

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

Freeman, P. 1987b. "*A Perspective on Reusability*". IEEE Tutorial: Software Reusability (ed. P. Freeman), IEEE Computer Society Press: 2-8.

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

Girow, Andrew. 1996b. "*Binary Relations Approach to Building Object Database Model*". Object Currents. Vol.1, Issue 11, SIGS Publications, November 1996.

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

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

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

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

McIlroy, M. D. 1976. "*Mass-produced Software Components*". In J.M. Buxton, P. Naur, and B. Randell, editors, Software Engineering Concepts and Techniques; Van Nostrand Reinhold (1968 NATO Conference on Software Engineering): 88-98.

Marqués Corral, José Manuel, Cuesta, Raúl, de la Fuente, Pablo. 1994. "*BLC++: A Logical Browser for C++*". En los proceedings of the Conference on Modelling and Simulation. Guasch and Huber, editors: 362-366.

Marqués Corral, José Manuel. "*Reutilización y Diseño Orientado al Objeto: Informe de Resultados y Propuestas de Investigación*". En las Actas de las I Jornadas de Trabajo en Ingeniería del Software (Sevilla, Noviembre 1996)

Maudes Raedo, Jesús Manuel, Marqués Corral, José Manuel, Hernández, Carmen, García Peñalvo, Francisco José. "*Versionado Total de Esquemas a través de Reglas ECA*". En las Actas de las II Jornadas de Investigación y Docencia en Bases de Datos (Madrid, Julio1997).

Metamodel. l997. "*Metamodelling Glossary*". http://www.metamodel.com/glossary.html.

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

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

Pastor, Oscar, Ramos, Isidro. 1995. "*Oasis 2.2: A Class-Definition Language to Model Information Systems Using an Object-Oriented Approach*". SPUPV-95.788, Universitat Politècnica de València.

Paton, Norman W., Diaz, Oscar, Barja, María L. 1993. "*Combining Active Rules and Metaclasses for Enhanced Extensibility in Object Oriented Systems*". Data & Knowledge Engineering. Vol. 10: 45-63. North-Holland.

Penadés, M. C., Ramos, Isidro, Canós, J. H. Tolsa, J., Ferrer, J. "*Análisis, Diseño e Implementación OO de un Entorno de Prototipación Automática*". En las Actas de las I Jornadas de Trabajo en Ingeniería del Software (Sevilla, Noviembre 1996).

Ramos, Isidro, Toval, Ambrosio. "*Entorno General de Producción Automática de Prototipos de Software Orientado a Objetos OOAP (Object-Oriented Assistant Prototyper)*". Actas de las I Jornadas de Informática y Automática (Puerto de la Cruz-Tenerife. Asociación Española de Informática y Automática (AEIA), Julio 1995).

Rational Software Corporation. 1997a. "*Unified Modeling Language. UML Summary*". Rational Software Corporation. Version 1.0. 13 January 1997.

Rational Software Corporation. 1997b. "*Unified Modeling Language. UML Semantics*". Rational Software Corporation. Version 1.0. 13 January 1997.

Rational Software Corporation. 1997c. "*Unified Modeling Language. UML Semantics Appendix M1 – UML Glossary*". Rational Software Corporation. Version 1.0. 13 January 1997.

Rational Software Corporation. 1997d. "*Unified Modeling Language. Notation Guide*". Rational Software Corporation. Version 1.0. 13 January 1997.

Roddick, John F. 1995. "*A survey of schema versioning issues for database systems*". Information and Software Technology, Vol. 37, Number 7.

Rumbaugh, James, Blaha, Michael, Premerlani, William, Eddy, Frederick, Lorensen, William. 1996. "*Modelado y Diseño Orientados a Objetos. Metodología OMT*". Prentice Hall.

Sparks, Steve, Benner, Kevin, Faris, Chris. 1996. "*Managing Object Oriented Framework Reuse* ". IEEE Computer. September 1996: 52-61.

Toval, Ambrosio, Fernández, J. Luis, Nicolás, Joaquín, Sáez, José, Heredia, Diego. "*Desarrollo del Soporte Metodológico OM: 'OASIS Method'*". En las Actas de las I Jornadas de Trabajo en Ingeniería del Software (Sevilla, Noviembre 1996).

Trump, Daniel. "*Using the WWW and the Internet to Support Corporate Reuse*". WISR8. Ohio. 1997.

Yourdon, E. 1997. "*Rational Software: The First Billion-Dólar Case Vendor?*". Application Development Strategies. Vol. 9. Nº1.