1. INTRODUCCIÓN Y OBJETIVOS

1.1.- Introducción general

La reutilización del software está en auge, puesto que es uno de los elementos que contribuyen a la producción de software más fiable, más barato y en un tiempo menor, en un campo muy competitivo como es el mercado de desarrollo de sistemas software.

La reutilización del software se puede dividir en dos grandes campos como son el desarrollo para reutilización y el desarrollo con reutilización. En el primer caso se pretende la construcción de elementos software tan genéricos que permitan tras un refinamiento, ser utilizables de nuevo en la construcción de otros elementos software nuevos. Es aquí donde se usa el segundo concepto: el desarrollo con reutilización. Así, durante la construcción de nuevos elementos software no sería necesario desarrollar componentes que se hayan construido antes, pudiéndose utilizar los elementos software genéricos proporcionados por el desarrollo para reutilización. Es por tanto en el desarrollo con reutilización donde se manifiesta que la reutilización del software puede abaratar la construcción de software eliminando el tiempo de desarrollo de componentes que ya hechos y probados.

Como solución al problema del almacenamiento de estos componentes software genéricos creados en el desarrollo para reutilización surgen los repositorios. Dado que la reutilización de productos o componentes software ya hechos, por su definición va estrechamente ligado al paradigma de la programación orientación al objeto, uno de los medios que pueden favorecer la expansión de la reutilización será la integración del proceso de reutilización el las herramientas de desarrollo de software orientado a objetos junto con el uso de repositorios.

1.2.- Trabajos previos

Dadas las ventajas de la reutilización del software, surgen investigaciones sobre como incorporarla a la producción de software de alta calidad y en tiempo reducido, mediante la creación de herramientas que automaticen el proceso de catalogación y extracción de elementos software reutilizables dentro de las propias herramientas de desarrollo software convencionales. Así nace el grupo GIRO (*Grupo de Investigación en Reutilización y Orientación al Objeto de la Universidad de Valladolid*), que para apoyar el uso e investigación de la reutilización crea un repositorio de elementos software reutilizables y el conjunto de herramientas que posibilitan darle uso al mismo. El grupo GIRO se encuentra compuesto por miembros del Departamento de Informática de la Universidad de Valladolid, el Área de Lenguajes y Sistemas Informáticos de Burgos y del Departamento de Informática y Automática de la Universidad de Salamanca

Para catalogar los elementos software reutilizables del repositorio, el grupo GIRO define una estructura de reutilización compleja, el Mecano, estructura que cuenta con elementos software de grano más fino como medida para que el proceso de reutilización cubra todos los niveles de abstracción. El repositorio creado almacenaría

los elementos reutilizables de grano fino, a los que llamaría Assets, en una base de datos ORACLE y permitiría su catalogación y recuperación vía WEB, estableciendo relaciones lógicas entre ellos para formar los elementos software reutilizables de grano grueso definidos anteriormente como mecanos.

Para la inserción de forma automática de elementos reutilizables en el repositorio creado se desarrolló una funcionalidad añadida a la herramienta CASE llamada TOGETHER (*PFC Mecanos representando proyectos Together - Inserción automática en el repositorio de GIRO*).

La herramienta TOGETHER era idónea para el uso de reutilización, puesto que cubre todos los niveles de abstracción y por tanto permite incorporar de forma sencilla el modelo de Mecano a su funcionamiento.

La herramienta creada dotaba a TOGETHER de la capacidad de insertar elementos software reutilizables construidos con ella en el repositorio creado por el grupo GIRO. De esta manera se lograba cubrir una de las facetas de la reutilización: el desarrollo para reutilización, uniéndolo al proceso de construcción software orientado al objeto. Además, almacenaba en el repositorio de GIRO las relaciones lógicas existentes entre estos elementos reutilizables (assets) posibilitando su extracción como un todo o un elemento reutilizable de grano más grueso (mecano).

Aun quedaba dotar a la herramienta Together de desarrollo con reutilización.

1.3.- Objetivos

1.3.1.- Objetivos de carácter funcional

El objetivo fundamental de este proyecto es dotar a la herramienta TOGETHER con la funcionalidad de desarrollo con reutilización. Esto junto a los trabajos previos expuestos anteriormente convertiría a TOGETHER en una herramienta de desarrollo software que además de cubrir por completo el ciclo de vida del software permitiera la reutilización y el uso intensivo del repositorio creado por el grupo GIRO.

La funcionalidad de la reutilización que queda por cubrir en TOGETHER es el desarrollo con reutilización, es decir la extracción de elementos reutilizables del repositorio GIRO y la incorporación a un proyecto software en TOGETHER que requiera dichos componentes. Por tanto los objetivos de carácter funcional a destacar serían:

- Ampliar la funcionalidad de la herramienta TOGETHER permitiéndole usar e incorporar a los proyectos software creados con esta herramienta los elementos reutilizables de grano fino (assets) o de grano grueso (mecanos).

- Ampliar la funcionalidad de la herramienta TOGETHER para que pueda comunicarse con un repositorio que responda al modelo de mecano y extraer de el los elementos reutilizables que el desarrollador con reutilización que use TOGETHER estime oportunos.

- Definir el modelo de intercambio de datos que tanto TOGETHER (cliente) como el repositorio (servidor) utilizarán a la hora de comunicarse entre sí.

1.3.2.- Objetivos de carácter técnico.

Los objetivos de carácter técnico de la herramienta a desarrollar serian:

- Intercambio vía Internet de datos entre el repositorio (servidor) y la herramienta TOGETHER (cliente). La comunicación por Internet dada la expansión mundial que se esta viviendo de la gran red de redes, posibilita el acceso de la herramienta TOGETHER al servidor desde cualquier localización, de forma rápida y segura.

- Limitación del acceso al repositorio a individuos no autorizados por los creadores o gestores del repositorio (en este caso GIRO). El concepto de repositorio puede ampliarse a repositorios software comerciales que requieran de una autorización expresa de sus gestores o creadores para la descarga de elementos reutilizables del repositorio, así como la trazabilidad de las acciones realizadas por los clientes de dicho repositorio.

- Proceso de desarrollo del software Orientado al Objeto. El proceso de desarrollo del software debe realizarse siguiendo las metodologías de creación del software actuales frente a las antiguas metodologías como la estructurada, puesto que la metodología orientada a objetos va íntimamente ligada con la reutilización de componentes software.

- Desarrollo en JAVA. La herramienta CASE TOGETHER esta completamente construida en Java y para dotarla de funcionalidades añadidas, éstas deben se desarrolladas en Java. Esto no limita en modo alguno las capacidades del proyecto a realizar, puesto que Java es un lenguaje de programación orientado a objetos, moderno y muy extendido a la hora de desarrollar aplicaciones de comunicaciones sobre Internet. Java además aporta una serie de características, herramientas ya construidas y soporte para las comunicaciones que lo hacen especialmente idóneo para este caso (Servlets, acceso a Bases de datos, criptografía y seguridad ...)

- Portabilidad del producto a cualquier plataforma. El producto construido debe ser funcionar en cualquier plataforma. Puesto que la herramienta se desarrollará en Java, que es un lenguaje de programación multiplataforma este objetivo queda cubierto.

- Utilización de DTDs (*Document Type Definition*) y XML (*eXtensible Markup Language*) para la lectura de propiedades de los elementos reutilizables del repositorio. Con XML se puede determinar una estructura fija para el intercambio de información de los elementos reutilizables del repositorio (propiedades) entre el servidor y el cliente. Esto unido al hecho de que en un trabajo previo fue construida una DTD que respondía al modelo de Mecano del repositorio, y al gran número de parsers XML para Java en la red, convierten a XML en el formato idóneo para realizar esta operación.

- Utilización de XMI (*XML Metadata Interchange*) para la representación física de los assets del repositorio. En los trabajos previos realizados, los elementos reutilizables eran almacenados con este formato en el repositorio, puesto que XMI era un estándar de OMG que representaba a UML y que usaba TOGETHER como modelo de intercambio de datos. Por tanto los assets en formato XMI deberán se importados de nuevo en los proyectos de TOGETHER cuando el desarrollador con reutilización los solicite.

- Diseñado para ser ampliable. La herramienta a desarrollar debe ser lo bastante abierta para que en un futuro cercano se la amplíen funcionalidades o se decida fusionar con la herramienta de desarrollo para reutilización construida con anterioridad. También las herramientas de terceros usadas deben ser fácilmente incorporadas a proyectos similares en el campo de la reutilización (parsers, acceso a BD, seguridad, etc).

2. LA REUTILIZACIÓN DEL SOFTWARE

El idea de la reutilización del software lleva años aplicándose pero sólo se llegaba su aplicación al campo de la implementación mediante el uso de bibliotecas de funciones ya programadas. Actualmente, el concepto de reutilización se ha extendido a todos los productos que se derivan del desarrollo software independientemente de su nivel de abstracción siempre que cumplan unos principios de genericidad.

Aquí se da una breve reseña histórica del nacimiento de la reutilización y los trabajos realizados en este campo por el grupo GIRO.

2.1 El Concepto de reutilización

El término reutilización vio la luz en la conferencia de la NATO de 1968 sobre Ingeniería del Software propuesto por M.D. McIlroy como una de las principales y mejores soluciones para aumentar la productividad de los desarrolladores de software, así como para aumentar la fiabilidad de los productos software construidos.

Se podría definir reutilización 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*" o como "*la utilización de elementos software existentes durante la construcción de un nuevo sistema software*".

Aún hoy en día la reutilización continúa siendo uno de los campos de mayor investigación dentro de la ingeniería del software ya que a pesar de sus bondades y de ser la solución mas realista para los problemas de productividad en el desarrollo, muchos autores afirman que no se han conseguido avances significativos para la utilización sistemática de la reutilización en el proceso de construcción del software.

La definición clara y precisa de qué es un elemento reutilizable se hace necesaria para el uso sistemático de la reutilización en el desarrollo software. Así nace el concepto de asset o elemento reutilizable, como "*cualquier componente que es específicamente desarrollado para ser utilizado, y es actualmente utilizado, en más de un contexto*" [Karlsson]. Dicho elemento reutilizable pertenecería a cualquier nivel de abstracción de un desarrollo que se considere reutilizable llegando desde los documentos de análisis al código fuente.

2.2 Actividades de la reutilización

Dado que el proceso de reutilización debe cubrir tanto desarrollar como reutilizar componentes software aparecen dos grandes actividades en las que se puede dividir la reutilización: el desarrollo para reutilización y el desarrollo con reutilización.

2.2.1 El desarrollo para reutilización

Esta actividad tiene por fin seleccionar y catalogar los componentes software propensos a ser reutilizables. Los principios en los que se debe basar el desarrollo de

componentes reutilizables debe ser la calidad y fiabilidad frente al coste o tiempo de producción de estos componentes Así mismo se debe verificar que dichos elementos satisfacen varios conjuntos de requisitos, condición necesaria para que estos sean reutilizables.

2.2.2 El desarrollo con reutilización

Esta actividad consiste en la generación de nuevos productos software a partir de los elementos reutilizables generados durante el desarrollo para reutilización. Esta actividad depende en gran medida de los componentes software construidos durante el desarrollo para reutilización y debe cubrir estos cuatro problemas que se plantean:

- Seleccionar y recuperar los componentes
- Comprender y evaluar los componentes
- Adaptar los componentes un producto software concreto
- Integrar los componentes al proyecto software en desarrollo

2.3 Repositorios y la automatización de la reutilización.

La reutilización sólo tiene sentido si existe una entidad que almacene y catalogue los elementos reutilizables construidos y que permita su recuperación. Es esta entidad, el repositorio, es el que enlazará las dos actividades de la reutilización: el desarrollo para reutilización y el desarrollo con reutilización.

El concepto de repositorio se basa por definición en una base de datos de información compartida sobre los elementos que se producen o se usan en un desarrollo software. Esta definición es difusa puesto que incluiría desde simples bases de datos a entornos totalmente automatizados para la inserción y extracción de elementos reutilizables en las propias herramientas de desarrollo software.

Sobre estos entornos recae la mejor oportunidad de que dispone la reutilización para imponerse en el campo del desarrollo del software. La existencia de un entorno completo de desarrollo para y con reutilización en una herramienta de desarrollo de software (y transparente de cara usuarios a los servicios proporcionados por un repositorio en concreto) podría conseguir que muchos desarrolladores se decantaran por la reutilización salvando las conocidas trabas con las que la reutilización se encuentra en la práctica (el síndrome no inventado aquí, el acceso a los componentes reutilizables, la estrategia de las compañías de software, etc...).

3. REUTILIZACIÓN EN GIRO

La labor de investigación del grupo GIRO para la reutilización se centró en la definición de unos modelos formales de elementos reutilizables que estuvieran orientados al objeto.

Los modelos formales definidos cubrirían tanto los elementos reutilizables de grano grueso y que abarcarían varios niveles de abstracción como los elementos de grano fino, así como una serie de relaciones que permitieran su utilización de forma automatizada en el proceso de desarrollo software. Este apartado es un resumen de los trabajos realizados por el grupo GIRO en este aspecto usando para los diagramas el lenguaje de modelado UML 1.1.

3.1 Asset

El concepto de asset nace a la hora de definir el elemento reutilizable de pequeña granularidad que se relaciona con otros elementos reutilizables y que puede pertenecer a cualquier nivel de abstracción.

Bajo estas premisas, se puede definir asset como aquel elemento software, que con independencia de su nivel de abstracción, ha sido seleccionado y preparado para su uso en desarrollos software diferentes al desarrollo software de origen.

Esta definición de asset implica lo siguiente:

- Se mantiene la extensión del alcance de la reutilización a todas las fases de un desarrollo software.
- Se incide en el hecho de que la reutilización requiere de un coste adicional en los proyectos software, coste que se refleja en el esfuerzo de diseño de los diferentes assets para su uso en otros proyectos software distintos al de su creación, o bien en el esfuerzo de su identificación, extracción y configuración de proyectos software ya terminados.

Los assets compartirían entre sí una serie de características que influirán en su futura capacidad de reutilización:

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

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

3.2 Mecano

El concepto de mecano surge respuesta al problema de definir una estructura compleja que cubra simultáneamente varios niveles de abstracción y que sirva de soporte a un elemento software reutilizable.

Por tanto, Mecano se define como, “un metamodelo de elementos o componentes de software reutilizables de diferente nivel de abstracción, Assets, y sus interdependencias”.

Dicha estructura debería cumplir los siguientes requisitos:

- Aumento del nivel de abstracción de la reutilización: Debe aumentarse el alcance de la reutilización del software, dirigiéndolo hacia niveles de mayor abstracción que la implementación.
- Estructura con soporte de diferentes niveles de abstracción de forma simultánea:
- La estructura debe acoger varios assets relacionados entre sí y clasificados en diferentes niveles de abstracción.
- Soporte para la trazabilidad: De forma que se pueda navegar por los diferentes assets a través de la red de enlaces existentes entre ellos, con especial interés en los enlaces entre los assets clasificados en diferentes niveles de abstracción.
- Definición de un soporte para el desarrollo para reutilización: La estructura debe dar soporte a los assets que se diseñen (o se descubran con técnicas de reingeniería) para su reutilización.
- Base para el desarrollo con reutilización: La estructura debe ofrecer una serie de facilidades que ofrezcan la flexibilidad necesaria para su reutilización en desarrollos futuros.

Bajo estas premisas se puede definir mecano como "*Un sistema de elementos software reutilizables, clasificados en diferentes niveles y relacionados entre sí, ya sea dentro de un mismo nivel de abstracción (relaciones intranivel) o entre diferentes niveles de abstracción (relaciones internivel), cumpliéndose la restricción de que debe existir al menos una relación internivel"*.

Con esta definición se tiene que un mecano puede abarcar un máximo de tres niveles de abstracción y un mínimo de dos, que en orden decreciente de abstracción serían el nivel de requisitos (análisis), el nivel de diseño y el nivel de implementación.

Esto implica que se obtienen cuatro configuraciones posibles de los mecanos en función de los niveles de abstracción que abarquen: **R-D-I**, **R-D**, **R-I** y **D-I**.

Al tratar al mecano como un conjunto de elementos software reutilizables de distinto nivel de abstracción se obtiene un esbozo de lo que será finalmente el modelo de mecano:

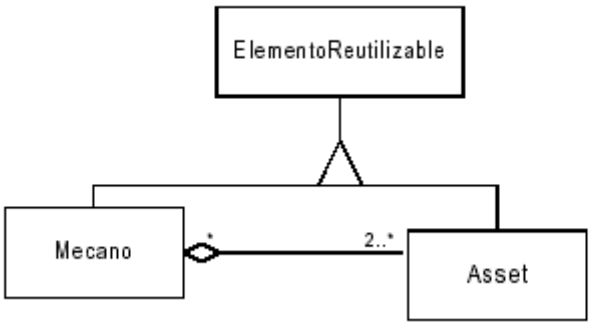

Figura 3.1. Modelo preliminar de un mecano.

3.3 El Modelo de Mecano

Una vez definidos los elementos reutilizables de grano fino (assets) y de grano grueso (mecanos) queda por definir un modelo que represente a ambos sobre el que basar el repositorio, así como definir los posibles tipos de relaciones que los assets pueden tener con otros assets para formar mecanos.

Antes de definir este modelo hay que tener en cuenta una serie de requisitos y restricciones que un estudio más detallado y la propia experiencia ponen de relieve a la hora de definir el modelo de mecano GIRO:

- Todo asset debe ser de un tipo predefinido, que indique de qué clase de artefacto software se trata. Cada uno de los assets componentes de un mecano debe
- pertenecer a un tipo de asset que recoja las características propias de ese tipo de elemento software y permita establecer un control sobre con que otros assets puede
- relacionarse.
- Se debe tener una flexibilidad para incrementar los tipos de assets soportados. El mundo del software es sumamente cambiante y evolutivo. Es frecuente la aparición
- de nuevos métodos de desarrollo software que utilizan nuevas técnicas, o variaciones de las existentes, para la generación de elementos software que pueden
- ser reutilizables.
- La estructura del asset debe dar soporte a la información lógica y física del asset, además de ofrecer detalles sobre la seguridad, la calidad y los aspectos
- administrativos del mismo. La información relacionada con un asset puede clasificarse en diferentes subsistemas de información. Por un lado están los atributos
- que capturan toda la información lógica del asset (información derivada del tipo de asset al que pertenece, referente a sus creadores, sobre las personas que pueden
- acceder a él, las métricas y criterios de certificación…) y por otro lado está la información que indica donde se encuentra ese asset físicamente (fichero, URL…).
- Un asset puede formar parte de varios. Un mismo asset puede formar parte de varios desarrollos (eso al menos es el objetivo de la reutilización), y por lo tanto puede aparecer como componente de distintos mecanos que conviven en el repositorio.
- Los enlaces semánticos entre assets deben ser de un tipo de relación: Cada uno de los enlaces entre los assets componentes de un mecano viene caracterizado por un
- tipo de relación semántica o relación de dominio.
- Las relaciones semánticas (relaciones del dominio) entre los assets se derivan del dominio del desarrollo del software. No debe perderse de vista que al definir una
- estructura de elemento software reutilizable, se está inmerso en el dominio del desarrollo de software, y por tanto no debe extrañar el hecho de que algunos de los

- tipos de relaciones semánticas que caracterizan los enlaces semánticos entre los assets coincidan en nombre con las relaciones estructurales necesarias para
- establecer el modelo.
- Los tipos de relaciones semánticas deben definir perfectamente las restricciones necesarias para evitar la introducción de incongruencias en la estructura de
- reutilización. Más concretamente, los tipos de relaciones semánticas deben establecer las restricciones oportunas para evitar incongruencias a la hora de enlazar
- tipos de assets, incorporando de esta forma una semántica que se echaba en falta en los modelos de elementos software reutilizables existentes.
- Un mecano puede estar formado por un conjunto de mecanos existentes. Como se ha expresado anteriormente, un mecano es un agregado de assets relacionados. Pero,
- también puede definirse en función de otros mecanos.
- Un mecano está ligado a uno o a varios dominios. La reutilización del software es una de las prácticas que mayores beneficios en productividad y calidad puede
- ofrecer en el campo del desarrollo de software. Pero, cuando la reutilización se conduce hacia una visión multinivel de la misma, se está caminando hacia una
- reutilización vertical en dominios definidos y precisos.
- Un mecano está ligado a un contexto de reutilización. El contexto de un mecano es el entorno en el que se le permite operar. Contiene las restricciones que han de
- cumplirse en el entorno en el que el mecano será reutilizado.
- Un mecano debe soportar la evolución de sus assets componentes. La modificación de un asset es una operación básica que debe darse en cualquier
- repositorio. Según como se realice esta modificación, se tendrá una modificación de un asset o una versión del mismo.

3.3.1. Estructura de los assets componentes de un mecano (asset GIRO)

Bajo las anteriores premisas se creó el asset GIRO basándose en el modelo BIDM [IEEE, 1995] utilizándose para su representación un modelo objeto.

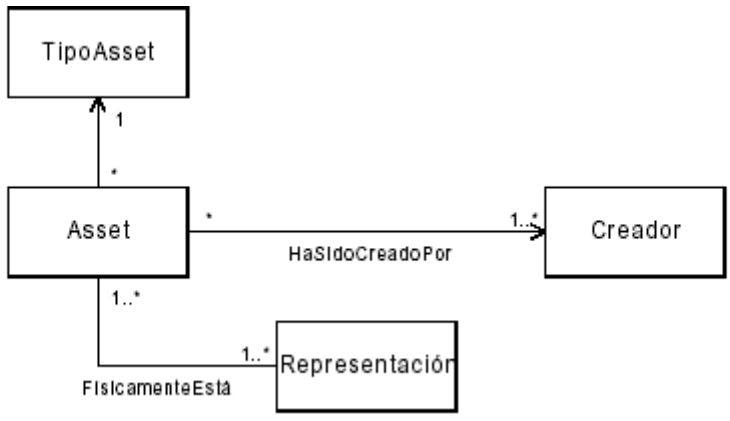

Figura 3.2. Modelo básico de Asset

La clase Asset sería el centro de este modelo representando un asset lógico genérico (sin ninguna restricción en cuento a su representación física) y que tendría los siguientes atributos:

- **Identificador**: Es un identificador único dentro del sistema.
- **Nombre**: Es el nombre o el título.
- **Resumen**: Explicación o definición general del asset.
- **FechaCreación**: Fecha de creación del asset.
- **FechaIntroducción**: Fecha en que el asset fue introducido en el repositorio.
- **FechaModificación**: Fecha de última modificación del asset.
- **Dominio(s)**: Área de aplicación, actividad o conocimiento del asset. (la información del dominio es más propia del mecano al ser este un elemento reutilizable de mayor entidad que sus assets componentes).
- **PalabrasClave**: Palabras o frases que describen al asset.
- **NivelAbstracción**: Fase del ciclo de vida en que fue generado el asset.
- **Método**: Método de desarrollo utilizado para generar el asset.
- **Entorno**: Entorno de destino del asset, esto es, plataforma hardware, sistema operativo y/o compilador necesarios para que el asset sea operativo.
- **Restricciones**: Información legal sobre el uso del asset, incluyendo copyright, patentes, derechos de explotación, limitaciones de exportación y licencias.
- **NivelSeguridad**: El nivel de seguridad más alto asignado a un asset o cualquier parte constituyente de un asset.
- **Coste**: Cuantía que debe pagarse por la reutilización del asset.
- **Idioma**: Lengua en la que se encuentra el asset.
- **Versión**: Designación de la versión de un asset.
- **NivelEvaluación**: Nivel de evaluación del asset establecido por el responsable del repositorio.

Por las restricciones al modelo de mecano antes expuestas se hace necesaria la clase TipoAsset para establecer un control sobre el tipo de relaciones que los assets pueden establecer entre ellos. La clase TipoAsset tiene como atributos:

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

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

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

- **Comentarios**: Posibles notas sobre el formato físico del asset.

La clase creador representa la organización o creador(es) del asset y de se mantenimiento. Sus atributos son:

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

El modelo básico de mecano visto en la Figura 3.1 queda ahora refinado y completado en el modelo básico de asset GIRO:

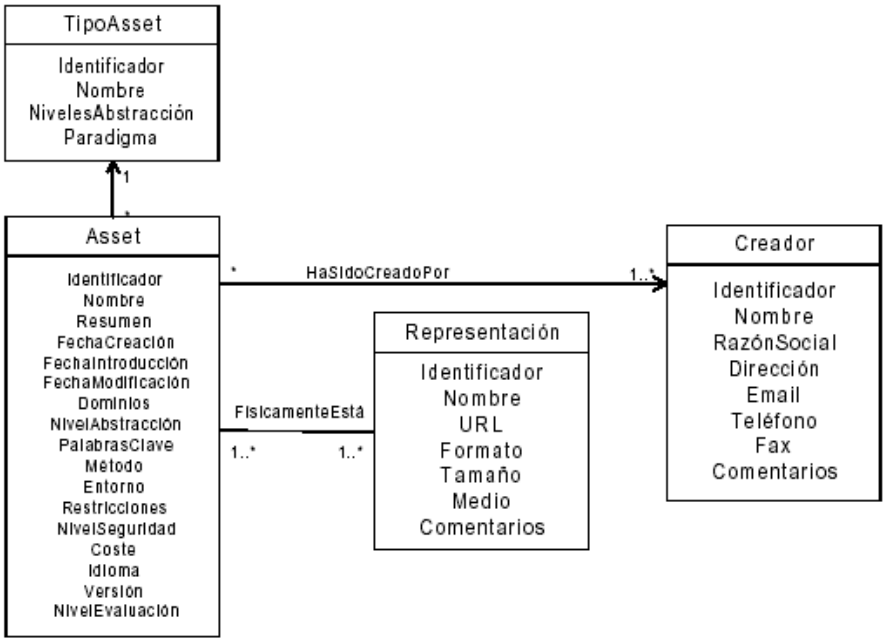

Figura 3.3. Modelo básico de asset GIRO

3.3.2. Relaciones entre los assets componentes de un mecano

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

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

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

3.3.3. Definición de las relaciones estructurales.

Dado que el mecano es una estructura compleja de reutilización que abarca varios niveles de abstracción es necesario definir y diferenciar las relaciones que se den entre assets del mismo nivel de abstracción (relaciones intranivel) y las relaciones que enlazan assets situados a distinto nivel se abstracción (internivel). Sus definiciones son:

Relaciones internivel:

- **Reificación**. Es una de las más importantes, puesto que da soporte a la semántica derivada del cambio de nivel de abstracción de un asset. Se define como: "Se dice que un asset Y reifica a un asset X cuando el asset Y se puede considerar como un refinamiento del asset X, y el asset X está clasificado en un nivel de mayor abstracción que el asset Y".

Relaciones intranivel:

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

- **Composición**: "Se dice que un asset Y está relacionado con un asset X mediante una relación de composición cuando el asset Y forma parte exclusivamente de la estructura de X, no teniendo sentido la reutilización del asset Y sin la reutilización del asset X, estando X e Y clasificados en el mismo nivel de abstracción" (tanto la agregación como la composición dan soporte a la existencia de relaciones entre assets de grano grueso y otros de grano fino en un mismo nivel de abstracción).

- **Usa a**: Esta relación representa a aquellas relaciones del tipo cliente-servidor, donde un asset solicita los servicios de otro asset. Su definición es: "Un asset Y usa un asset X cuando el asset Y depende del comportamiento especificado por el asset X para su definición o para completar su comportamiento, estando X e Y clasificados en el mismo nivel de abstracción".

- **Extensión**: Esta relación enlaza a aquellos assets cuya definición se da en base a la definición de otro asset, al que especializa. Se define como: "Un asset Y

extiende otro asset X cuando Y está definido en función de la definición de X, estando X e Y clasificados en el mismo nivel de abstracción".

- **Asociación**: Este tipo de relación engloba a todas las relaciones cuya semántica no se encuentra representada en las anteriores. Su definición es: "Un asset X está asociado con un asset Y cuando existe un enlace semántico bidireccional entre ambos assets, estando X e Y clasificados en el mismo nivel de abstracción".

3.3.4. Restricciones de las relaciones

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

3.3.4.1 Orden de las relaciones

Como puede deducirse de las definiciones de las relaciones estructurales enunciadas en el subapartado anterior, para el modelado de éstas se ha optado por la utilización de relaciones binarias.

Este hecho aparte de añadir naturalidad y sencillez a las mismas aporta trazabilidad de unos assets a otros, puesto que aporta la ventaja de que a la hora de reutilizar un asset también se puedan obtaner de forma fácil y rápida todos los assets relacionados con él y de forma recursiva todos los assets relacionados con estos últimos. Además, esta restricción no conlleva pérdida de semántica, ya que cualquier relación de orden superior a dos se puede modelar como un conjunto de relaciones binarias.

3.3.4.2. Dirección de las relaciones

Las relaciones estructurales identificadas en el espacio de reutilización se definen entre un origen y un destino, esto implica que aunque los enlaces semánticos entre los assets que forman los Mecanos puedan ser recorridos en ambas direcciones, la 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 poseen una polaridad semántica que se recoge en la tabla, a excepción de la asociación, considerada bidireccional por definición al estilo de Booch de UML.

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

Tabla 3.1. Relaciones estructurales con una polaridad semántica definida.

3.3.4.3. Carácter de las relaciones.

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

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

Estas definiciones de enlace fuerte y débil tienen una especial importancia en el caso de las relaciones de reificación y agregación puesto que:

- En el caso de la reificación se está asegurando la generación de mecanos en tiempo de reutilización como respuesta a las peticiones de los desarrolladores con reutilización, dado que bajo esta perspectiva generativa, la extracción del repositorio de un asset clasificado en un nivel de abstracción superior al de implementación siempre se verá acompañada de la extracción de los assets clasificados en niveles de abstracción menores relacionados con él.

- Para la agregación se tiene que ésta 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.

En la siguiente tabla se puede apreciar el carácter de las relaciones estructurales definidas para el modelo de mecano:

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

Tabla 3.2. Carácter de las relaciones estructurales en el proceso de generación de mecanos.

3.3.5. Relaciones en el modelo de mecano

Una vez definidas las relaciones entre assets así como sus restricciones, se puede proceder a su modelado dentro del contexto del modelo de Mecano GIRO. En la siguiente figura, la parte del modelo de mecano referido a las relaciones queda totalmente definida:

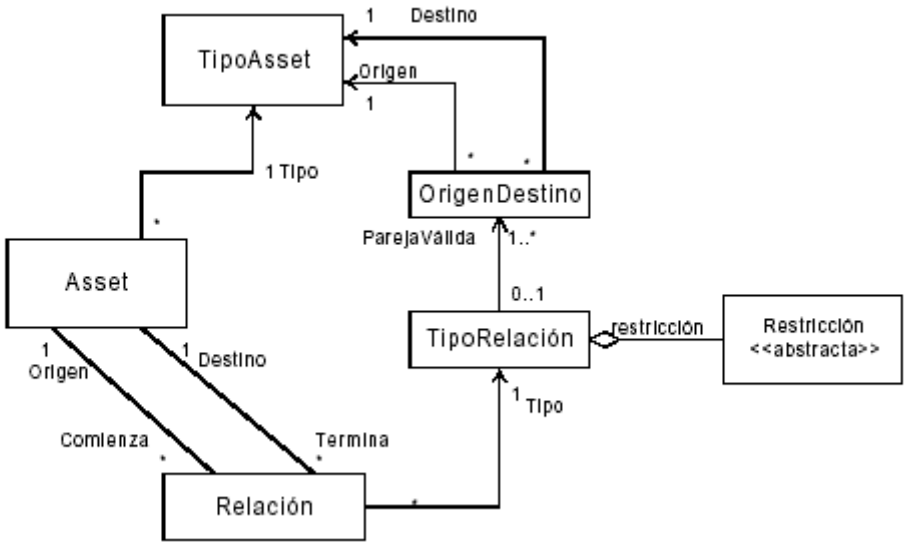

Figura 3.4. Modelado de las relaciones en el modelo de mecano.

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

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

Para el chequeo de las restricciones el método de construcción de la clase Relación delega en el objeto instancia de la clase TipoRelación cuya referencia ha recibido. La clase TipoRelación contiene dos atributos:

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

La clase TipoRelación tiene una asociación con la clase OrigenDestino y que queda expresado con :

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

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
```

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

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

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

3.3.6. Conclusiones sobre el modelo de mecano

Después del estudio realizado sobre la estructura de los mecanos, falta la unificación de todas las partes en lo que es el modelo de mecano:

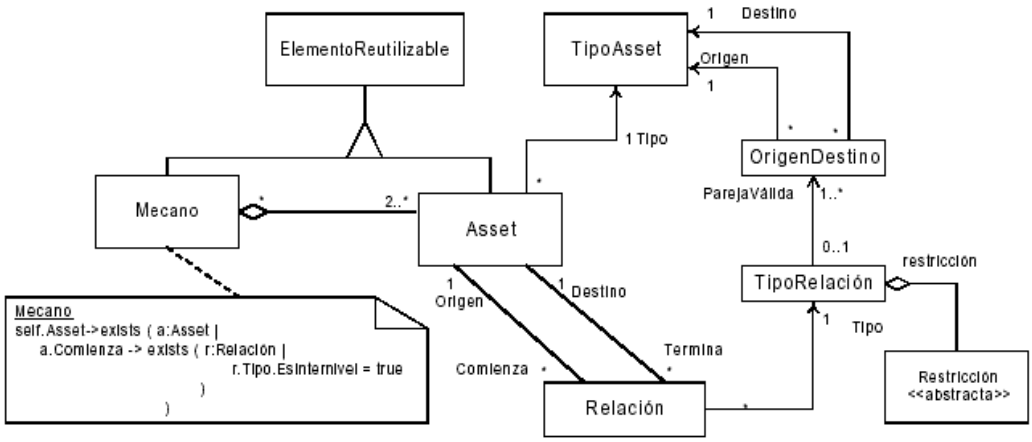

Figura 3.5. Modelo de Mecano

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

- Se ha utilizado un modelo objeto para representar las entidades semánticas del problema, utilizando UML 1.1 como lenguaje de modelado.
- En el modelo aparece la clase Mecano, que representa el elemento reutilizable complejo formado por una agregación de assets. Esta clase tiene como atributos un nombre y una lista de dominios en los que su reutilización es útil.
- Que un mecano sea una agregación y no una composición es fácil de argumentar. Cada uno de los componentes de un mecano puede ser parte de otros mecanos. Además, la eliminación de un mecano como elemento reutilizable no conlleva obligatoriamente a la eliminación de sus componentes.

- De los dos puntos anteriores se puede concluir que la connotación semántica de la relación de agregación entre el Mecano y los Assets se corresponde con lo que UML 1.1 denomina agregación compartida, y que no es más que un tipo de agregación débil que implica que si se destruye el todo de la agregación (el mecano) no se tienen que eliminar las partes constituyentes (los assets), y lo que es más importante, las partes constituyentes pueden aparecer en varios agregados al mismo tiempo
- El mecano presenta una restricción de definición importante que debe quedar reflejada en el modelo: 'todo mecano debe contar con al menos una relación internivel'. Esta restricción aparece en el modelo como una sentencia OCL en una nota asociada a la clase semántica Mecano.
- Se ha dado por sentado a lo largo de todo el documento que el repositorio es el soporte necesario para el almacenamiento de los assets producidos como consecuencia de los desarrollos para reutilización siendo, además, el soporte al desarrollo de sistemas soportando reutilización.
- El objetivo que se persigue con la definición de esta estructura no es el de construir una herramienta CASE, ni un motor de repositorios, sino que desde la perspectiva de los repositorios, definir un esquema que permita almacenar, manejar y recuperar elementos reutilizables, más concretamente mecanos.
- Con el modelo generado se tienen prácticamente las mismas posibilidades que ofrecen los repositorios más utilizados, algunos de los cuales han sido objeto de estudio en el presente trabajo. Incluso en el área de las relaciones semánticas se ha establecido las bases para un modelo más rico.
- Si un mecano se ve como el modelado conceptual de una estructura de información formada por assets y sus relaciones, el mecano está al mismo nivel que un esquema de bases de datos, siendo el sistema gestor de bases de datos el repositorio. Bajo esta perspectiva, el nivel de datos se corresponde con los assets introducidos en el repositorio, siendo el nivel de modelado el modelo de mecano, habiéndose utilizado como lenguaje de modelado UML 1.1.
- El modelo de mecano presenta una flexibilidad total para la inclusión de nuevos tipos de assets y de relaciones semánticas. Sin embargo, el modelo está cerrado en cuanto al tipo de relaciones estructurales soportadas.

4. PLAN DE DESARROLLO SOFTWARE

Todo sistema software mínimamente complejo requiere de una planificación en su desarrollo a fin de evitar riesgos innecesarios en la etapa de desarrollo.

El plan de desarrollo comienza con el estudio de la funcionalidad a cubrir, en base la los trabajos anteriores realizados por el *PFC Mecanos representando proyectos Together - Inserción automática en el repositorio de GIRO*, a los que este sistema software a desarrollar complementa y completa.

Era necesario la familiarización con las herramientas a utilizar, impuestas por las soluciones tomadas en los trabajos previos mencionados. Una vez realizado esto se debía llevar a cabo un estudio de las necesidades a las que este sistema software debía responder.

A partir de entones se hizo necesaria la adopción de la arquitectura Cliente-Servidor como la arquitectura a desarrollar. A partir de este punto se podrá comenzar el análisis para conocer que debe hacer el sistema y que tareas han de recaer sobre los integrantes del mismo.

También se optó desde el primer momento por un ciclo de vida iterativo, que a partir de unos prototipos de cliente y servidor obtenidos de un análisis, desarrollo e implementación iniciales, permitieran mediante la iteración de estos procesos obtener un sistema software fiable que cubre dolas las funcionalidades exigidas.

En concreto, para el caso del cliente y el servidor, a partir de un prototipo básico y un modelo de comunicaciones proporcionado por un análisis y diseño iniciales, se fue completando una a una y de forma pareja las funcionalidades (petición-respuesta) a la que ambos respondían, hasta llegar al producto final que en esta memoria se presenta.

5. DOCUMENTO DE REQUISITOS

5.1. Introducción

En este documento se detallan los requisitos que el sistema software a construir debe cumplir, siguiendo las directrices de UML Estos serán los requisitos necesarios para obtener un sistema fiable y correcto que se encargue de la extracción e incorporación a una herramienta CASE (Together) los elementos reutilizables de grano fino (assets) y de grano grueso (mecanos) presentes en el repositorio GIRO.

Para la definición de todos los componentes de este documento se tendrá en cuenta el uso de plantillas a fin de presentar cada elemento de forma detallada y de fácil lectura.

5.2. Definición de requisitos de carácter funcional

Número	rq-1.01
Descripción	Ampliar la funcionalidad de la herramienta CASE Together para permitir la inserción en un proyecto de dicha herramienta de elementos reutilizables tanto de grano fino (assets) como de grano grueso (mecanos). Creación de un módulo para la herramienta Together.
Prioridad	Muy Alta
Justificación	Funcionalidad del Sistema
Dependencias	rq-1.02

Número	rq-1.02
Descripción	Ampliar la funcionalidad de la herramienta CASE Together para permitir la conexión de ésta a un repositorio que responda al modelo de mecano y la extracción automática en el repositorio de elementos software reutilizables a petición de la herramienta CASE Together.
Prioridad	Muy Alta
Justificación	Funcionalidad del Sistema
Dependencias	rq-1.01

Número	rq-1.03
Descripción	Determinar el modelo de intercambio de información entre el repositorio (servidor) y la herramienta CASE (cliente) en las estructuras de envío de assets y mecanos.
Prioridad	Muy Alta
Justificación	Funcionalidad del Sistema
Dependencias	rq-1.02

Número	rq-1.04
Descripción	Determinar el modelo de intercambio de información entre el repositorio (servidor) y la herramienta CASE (cliente) en las estructuras de información lógica (propiedades) y física de los assets y los mecanos
Prioridad	Muy Alta
Justificación	Funcionalidad del Sistema
Dependencias	rq-1.02

Número	rq-1.05
Descripción	La nueva funcionalidad de la herramienta Together estará integrada totalmente en la interfaz gráfica de Together para facilitar al usuario la inserción de elementos reutilizables en sus proyectos.
Prioridad	Media
Justificación	Facilidad de manejo de la aplicación
Dependencias	rq-1.01, rq-1.06, rq-1.07, rq-1.08, rq-1.11

Número	rq-1.06
Descripción	El usuario podrá buscar los elementos reutilizables del repositorio que se adapten a sus necesidades a través de la herramienta Together.
Prioridad	Muy Alta
Justificación	Funcionalidad del Sistema
Dependencias	rq-1.01

Número	rq-1.07
Descripción	El usuario podrá seleccionar que elemento reutilizable quiere incluir a su proyecto de una lista que le suministrará la herramienta Together en base a sus necesidades.
Prioridad	Muy Alta
Justificación	Funcionalidad del Sistema
Dependencias	rq-1.01

Número	rq-1.08
Descripción	El usuario podrá examinar las características básicas (nombre, versión, autores, etc.) de un elemento reutilizable antes de decidirse a incluirlo en su proyecto.
Prioridad	Muy Alta
Justificación	Funcionalidad del Sistema
Dependencias	rq-1.01

Número	rq-1.09
Descripción	La inserción de un elemento reutilizable en un proyecto de la herramienta CASE Together también insertará todos los elementos reutilizables que le primero necesite para su reutilización.
Prioridad	Muy Alta
Justificación	Funcionalidad del Sistema
Dependencias	rq-1.01, rq-1.02

Número	rq-1.10
Descripción	El procesamiento de la información se realizará de forma transparente al usuario, solicitándole acciones sólo cuando la funcionalidad antes descrita lo requiera.
Prioridad	Media
Justificación	Facilidad de manejo de la aplicación
Dependencias	rq-1.01, rq-1.05

Número	rq-1.11
Descripción	Limitación de acceso a la funcionalidad creada para Together a personas no autorizadas, solicitando identificación (*login* y *password*) a todo usuario que desee utilizar la nueva funcionalidad creada para la herramienta Together.
Prioridad	Alta
Justificación	El acceso al repositorio y sus elementos debe controlarse para impedir que personas ajenas a la organización que gestiona el repositorio y que no tienen autorización de los gestores utilicen ese repositorio.
Dependencias	rq-1.02

Número	rq-1.12
Descripción	Simplificar la identificación de los usuarios no requiriendo ante cada acción su identificación
Prioridad	Media
Justificación	No se debe requerir ante cada acción que el usuario se identifique puesto que esto haría tedioso el manejo de la aplicación
Dependencias	rq-1.05, rq-1.10

5.3. Definición de requisitos de carácter no funcional

Número	rq-2.01
Descripción	Permitir la reutilización del software
Prioridad	Muy Alta
Justificación	El objetivo primordial de este proyecto y de todos los relacionados con él sigue siendo satisfacer a un mercado que exige productos y servicios cada vez más fiables, baratos y entregados a tiempo.
Dependencias	No tiene

Número	rq-2.02
Descripción	Intercambio vía Internet entre cliente y repositorio. Sistema distribuido, arquitectura Cliente-Servidor.
Prioridad	Muy Alta
Justificación	El repositorio de componentes software debe facilitar el acceso a cualquier desarrollador para y con reutilización de cualquier punto del globo. Dada la actual expansión de *Internet*, ésta es la mejor solución para posibilitar el intercambio de información de una forma rápida y eficiente.

Dependencias	No tiene

Número	Rq-2.03
Descripción	Asegurar la privacidad de la identificación (*login* y *password*) que el usuario suministra al sistema. Uso de algoritmos de criptografía basados en clave pública para garantizar la privacidad de la identificación del usuario
Prioridad	Alta
Justificación	Dado que se requiere comunicación vía Internet ente el cliente (Together) y el servidor (repositorio) y que desde el punto de vista de la seguridad se considera a Internet como canal de comunicación no seguro, es necesario impedir que terceras personas obtengan de manera fraudulenta una identificación (*login* y *password*) que les permita acceder al repositorio como usuarios de pleno derecho. Los algoritmos de criptografía basados en clave pública permiten que sólo el servidor pueda leer la identificación que un usuario le envía a través de Internet. A esto hay que añadir la disponibilidad hay de las herramientas de criptografía para cualquier plataforma y lenguaje de programación a través de Internet.
Dependencias	rq.-2.02

Número	rq-2.04
Descripción	Uso de servlets en la construcción del servidor para posibilitar el acceso a la base de datos del repositorio.
Prioridad	Alta
Justificación	Debido a su difusión en la red, y las grandes ventajas que ofrecen como por ejemplo, independencia de la máquina servidor y el sistema operativo en el que se ejecute, permitir el acceso a BD de manera segura... Para este proyecto se utilizará *Jakarta-Tomcat*, como programa servidor capaz de recibir peticiones a través de red y ejecutar el servlet adecuado para atenderlas.
Dependencias	rq.- 2.02

Número	rq-2.05
Descripción	Proceso de desarrollo Orientado al Objeto
Prioridad	Muy Alta
Justificación	El proceso de desarrollo del software debe ser realizado desde el punto de vista de nuevas metodologías orientadas a objeto frente a metodologías más viejas como metodologías estructuradas. Además el proceso de reutilización software está íntimamente ligado (aunque no es exclusivo) con las metodologías orientadas al objeto.
Dependencias	rq.- 2.01

Número	rq-2.06
Descripción	Desarrollo en Java
Prioridad	Media
Justificación	Puesto que se debe ampliar la funcionalidad de *Together* mediante un módulo surge la restricción de crearlo en Java, lenguaje de construcción de la propia herramienta CASE, y lenguaje en el que

	deben ser escritos sus módulos. Esta restricción no hace perder propiedades al proceso de desarrollo del proyecto puesto que hoy en día Java es un lenguaje consolidado y muy difundido, que además de sus características innatas (multiplataforma, orientado al objeto, seguridad) da soporte para las comunicaciones, el acceso a Bases de Datos, criptografía, etc...
Dependencias	rq.- 2.01, rq.- 2.02, rq.- 2.03, rq.-2.04, rq.- 2.05

Número	rq-2.07
Descripción	Independencia de plataforma del sistema software creado.
Prioridad	Media
Justificación	El sistema software creado debe poder ser implantado en cualquier máquina independientemente de su plataforma o sistema operativo. La elección de Java como lenguaje de programación cubre este requisito
Dependencias	rq.- 2.04, rq.- 2.05, rq.- 2.06

Número	rq-2.08
Descripción	Adaptabilidad del sistema a futuras ampliaciones o fusiones con otros sistemas similares
Prioridad	Alta
Justificación	El sistema debe ser lo suficientemente abierto para que en el futuro se le puede ampliar la funcionalidad para hacer frente a nuevos requisitos o para fusionarlo con otros sistema software dedicados a la reutilización como el presente proyecto.
Dependencias	rq.- 2.01, rq.- 2.07

Requisitos no funcionales impuestos por los trabajos previos a este proyecto:

Número	rq-2.09
Descripción	Utilización de DTDs (*Document Type Definition*) y XML (*eXtensive Mark-up Language*) para la representación lógica de los elementos reutilizables (propiedades y características de los mismos).
Prioridad	Alta
Justificación	La información lógica de los elementos reutilizables que se envía desde el servidor al cliente a fin de que el usuario la visualice debe estar estructurada. En trabajos anteriores de definió una DTD que determinaba la estructura de un documento XML que respondiera al modelo de mecano del repositorio GIRO. Se ha optado por el uso de esta DTD a fin que el intercambio de información lógica entre el repositorio GIRO y los clientes presentes y futuros de dicho repositorio sea uniforme y común para todos. Además, la gran variedad de parsers XML disponibles para todos los lenguajes de programación aportan interés a esta solución adoptada.
Dependencias	rq.- 2.02, rq.-2.07, rq.- 1.04, rq.- 1.06, rq.- 1.07, rq.- 1.08

Número	rq-2.10
Descripción	Utilización de XMI para la representación física de los elementos reutilizables y permitir el uso de otros formatos en casos puntuales.
Prioridad	Alta
Justificación	Los trabajos anteriores a este proyecto almacenaban en el repositorio de GIRO los assets en formato XMI, o texto plano si era código fuente. También existían assets en el repositorio con formato RATIONAL ROSE (MDL, texto plano). La propia funcionalidad del sistema impone la recuperación de los assets del repositorio en formato XMI (formato de intercambio de datos de Together como estándar de OMG, y formato de muchos assets del repositorio) y texto plano (códigos fuente), pero también sería interesante y productivo que la aplicación a desarrollar también permitiera la inserción de otros formatos de asset presentes en el repositorio.
Dependencias	rq.-2.01, rq.- 1.01

Número	rq-2.11
Descripción	El Gestor de Bases de datos será **Oracle8i**.
Prioridad	Alta
Justificación	El repositorio se apoya en una base de datos Oracle8i desarrollada en un proyecto anterior. Es necesario el uso de este gestor para poder acceder a los elementos del repositorio.
Dependencias	rq.-1.02, rq.-2.01

5.4. Definición de los actores

Un actor representa un conjunto coherente de roles que los usuarios de una entidad juegan cuando interactúan con ésta. Un actor tiene un rol por cada caso de uso relacionado con él. Una instancia de un actor representa una interacción individual con el sistema de una forma específica.

Se ha de identificar los distintos roles que los usuarios del sistema desempeñan desde la perspectiva del lenguaje de modelado usando UML.

En el sistema cliente se identificaron los siguientes actores:

- **Desarrollador con Reutilización**: es decir el usuario o usuarios de la herramienta CASE Together que desean insertar elementos software reutilizables en sus proyectos bajo Together. Será el actor el que inicie o solicite las distintas funcionalidades de esta aplicación.

En el cliente no tiene sentido el rol de Desarrollador para Reutilización, puesto que el desarrollo para reutilización no entra dentro de los objetivos de este proyecto ya que quedo este apartado cubierto con un trabajo anterior. Para futuras versiones cuyos objetivos sean la fusión de ambas funcionalidades (desarrollo para y con reutilización)

si será necesaria la presencia de ambos roles (Desarrollo para y con reutilización) en la identificación de los actores del sistema.

En los subsistemas cliente y servidor se identifican estos actores:

- **Herramienta CASE (Together)**: Es la propia herramienta CASE la que cargará el módulo desarrollado en este sistema, y será ella la que se comunique con el servidor para solicitar las extracciones de elementos reutilizables. Cualquier otro cliente distinto a Together pero que siguiera el mismo protocolo de comunicación con el servidor podría usar sus servicios.

5.5. Diagramas de Casos de Uso

Un caso de uso es una funcionalidad coherente de un sistema, subsistema o clase manifestada como una secuencia de mensajes intercambiados entre el sistema y uno o más actores que juntos establecerán las acciones o resultados observables que ejecutará el sistema. A continuación aparecerán los diagramas de los casos de uso del cliente y del servidor.

Desde el punto de vista del cliente aparecen las funcionalidades de identificación ante el sistema, cierre de sesión en el sistema, configuración de la dirección del servidor al que el cliente se conectará y la funcionalidad propia de este proyecto: la búsqueda, selección y extracción de elementos reutilizables del repositorio y su inserción en un proyecto en Together, tanto en el caso de elementos software reutilizables de grano grueso (mecanos) como de grano fino (assets). Esto puede observarse en la figura 5.1.

Desde el punto de vista del servidor las funcionalidades a cubrir por el sistemas serían la identificación ante el sistema de un cliente remoto, la búsqueda de un elemento software reutilizable en el repositorio, así como su extracción sea este un elemento reutilizable de grano grueso (mecano) como de grano fino (asset). Esto puede aprciarse en la figura 5.2.

El comportamiento de un Caso de Uso se puede especificar describiendo un flujo de eventos de forma textual, con texto estructurado formal o pseudocódigo, lo suficientemente claro para que alguien ajeno al sistema lo entienda fácilmente. Para la descripción mas detallada de los casos de uso así como de la mayoría de documentos en esta memoria se ha optado por el uso de plantillas

.

5.5.1. Diagrama de Casos de Uso : CLIENTE

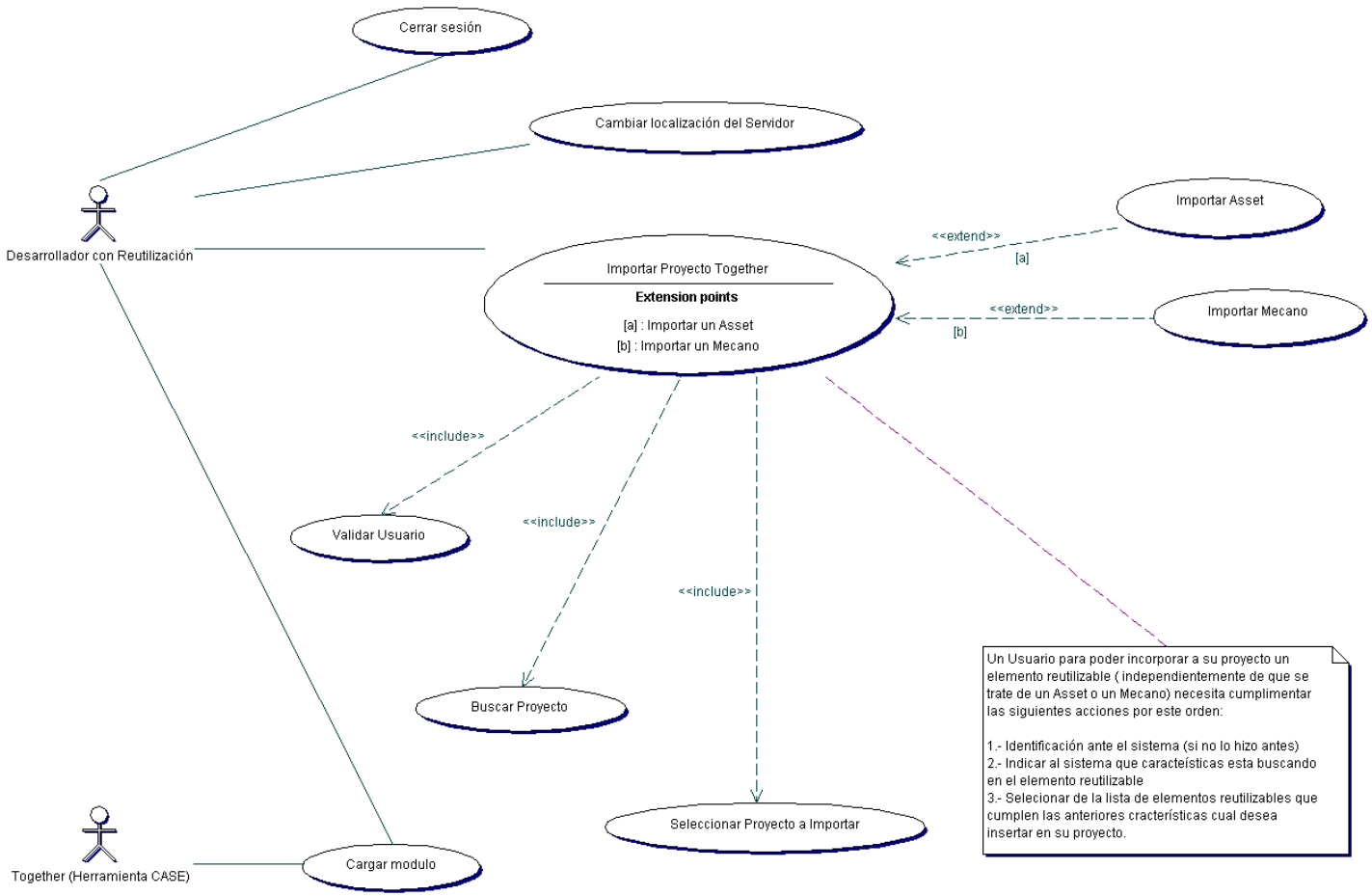

Figura 5.1.- Diagrama de Casos de Uso: Cliente

5.5.1. Diagrama de Casos de Uso : SERVIDOR

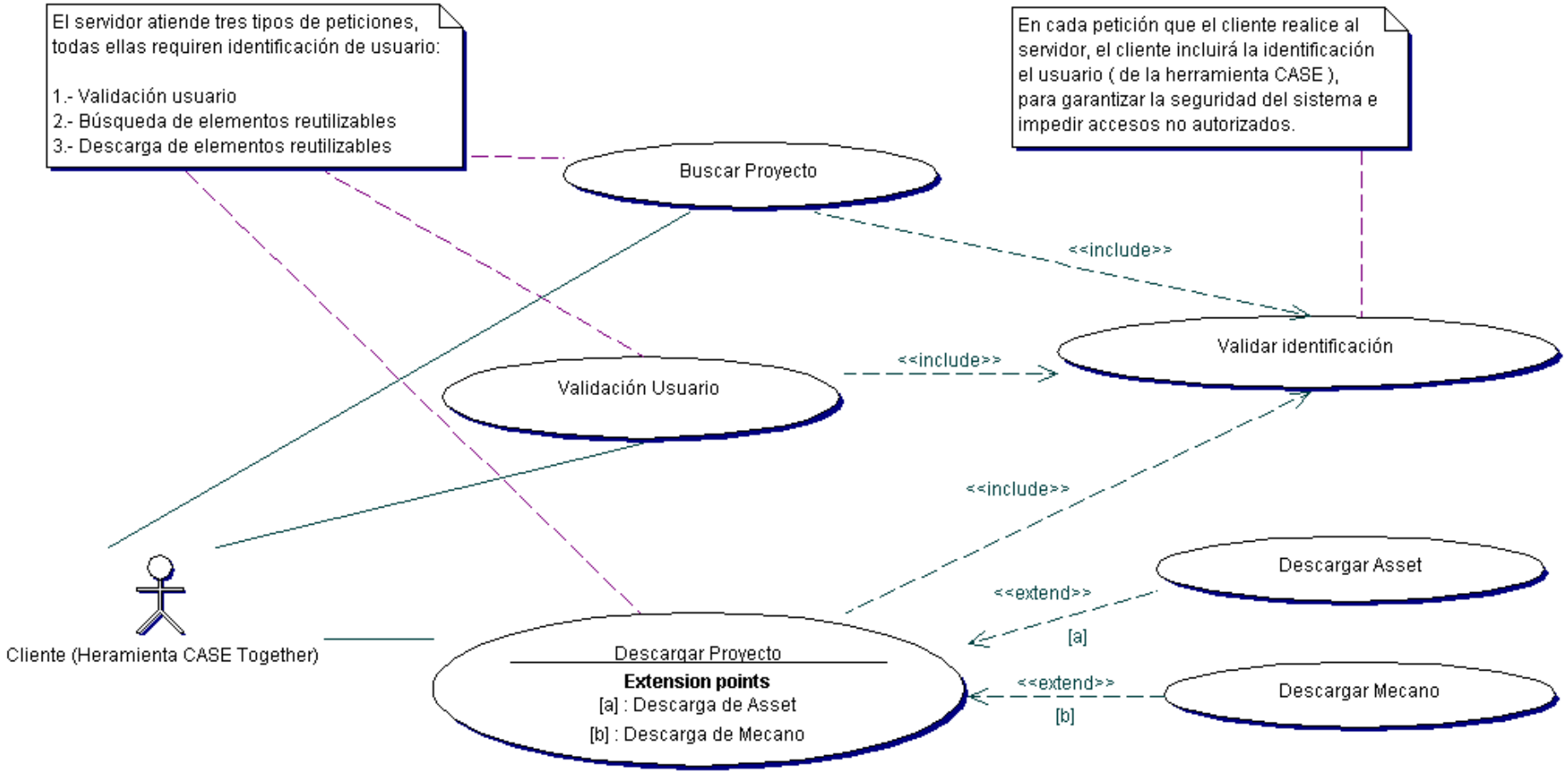

Figura 5.2.- Diagrama de Casos de Uso: SERVIDOR

5.5.3 Caso de Uso: Cargar Módulo

Mediante este caso de uso se inicia la aplicación de este proyecto (del lado del cliente). Pude iniciarse de dos formas, al arrancar la herramienta Together ésta la carga o bien el usuario explícitamente puede iniciarla,

Caso de uso	CS-1.01 Cargar módulo.	
Actores	Desarrollador con reutilización, *Together* (Herramienta CASE).	
Descripción	Mediante este Caso de Uso se consigue iniciar el módulo creado y configurarlo para poder mostrar las diferentes funcionalidades al usuario.	
Precondiciones	Ninguna.	
Secuencia normal	Paso	Acción
Opción 1	1	El usuario ejecuta la aplicación *Together* y ésta inicia el módulo aplicación.
Opción 1	2	Se generan los distintos menús que se presentará al usuario dentro de *Together.*
Opción 1	3	Se realiza la petición de la clave pública de encriptado y su autentificación.
Opción 2	1	Una vez iniciada la aplicación *Together*, el usuario inicia el módulo a partir de la pestaña de módulos de *Together*.
Opción 2	2	Se generan los distintos menús que se presentará al usuario dentro de *Together.*
Opción 2	3	Se realiza la petición de la clave pública de encriptado y su autentificación.
Escenarios secundarios	3	La localización inicial no es correcta, el servidor está inactivo o alguna entidad ajena al sistema intenta suplantar al servidor enviando otra clave pública. No se consigue la clave pública de encriptado y se informa de la situación al usuario para que cambie la localización o vuelva a intentar conseguir una nueva conexión.
Postcondiciones	Se ha iniciado el módulo aplicación informando al usuario del éxito en la comunicación con el servidor, así como de su localización.	
Comentarios	La clave pública de encriptado o el establecer otra localización del servidor se puede hacer con CS-1.04.	

El escenario correspondiente a este Caso de Uso sería el siguiente:

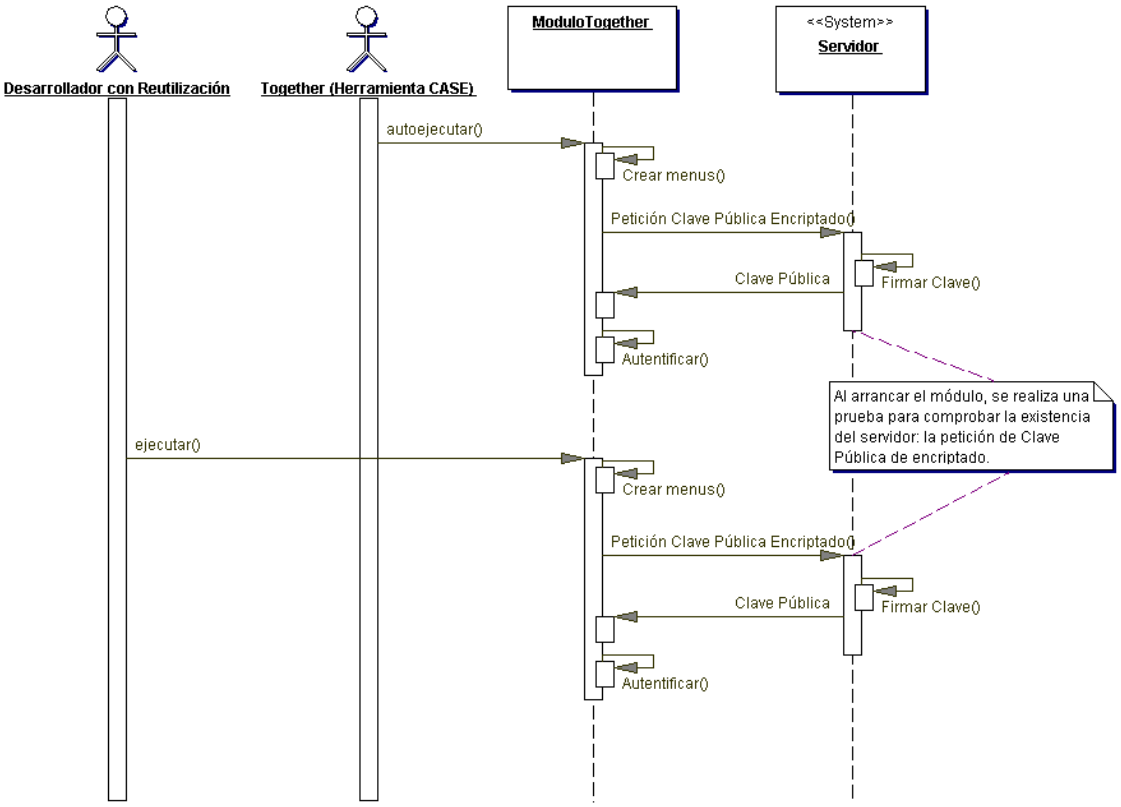

Figura 5.3.- Escenario: Cargar Módulo

5.5.4. Caso de Uso: Cambiar localización del servidor

Con este Caso de Uso se puede cambiar la dirección del servidor al que la herramienta CASE se conecta. El formato de la nueva dirección del servidor ha de ser completo, indicando protocolo, dirección y puerto de conexión.

Caso de uso	CS-1.02 Cambiar localización servidor	
Actores	Desarrollador con reutilización.	
Descripción	Mediante este Caso de Uso se consigue cambiar la localización o dirección que por defecto se establece para el servidor, sin tener que recompilar de nuevo el módulo o aplicación. Si la localización es cambiada se solicitará la calve pública de encriptado de identificación al nuevo servidor.	
Precondiciones	Ninguna.	
Secuencia normal	Paso	Acción
	1	El actor o usuario pulsa la opción *Cambiar de Servidor*.
	2	Solicitud en ventana de diálogo de la nueva localización.

	3	El usuario introduce los datos y pulsa el botón OK..
	4	El sistema cambia a la nueva localización.
	5	Con la nueva localización del servidor se realiza la petición de la clave pública de encriptado y su autentificación.
Escenarios secundarios	2	El cliente cancela la petición de datos (botón CANCEL), se va al paso 5 y se realiza la prueba de conexión descrita en ese paso.
	5	Si la nueva localización no es correcta, el servidor está inactivo o una entidad externa manipula la clave pública del servidor se informará al usuario del error por si desea de reintentar la operación.
Postcondiciones	1.- La localización vieja ha sido sustituida por la nueva. 2.- Si hubo éxito en la conexión, se obtendrá una clave pública de encriptado actualizada.	

El escenario para este Caso de Uso es:

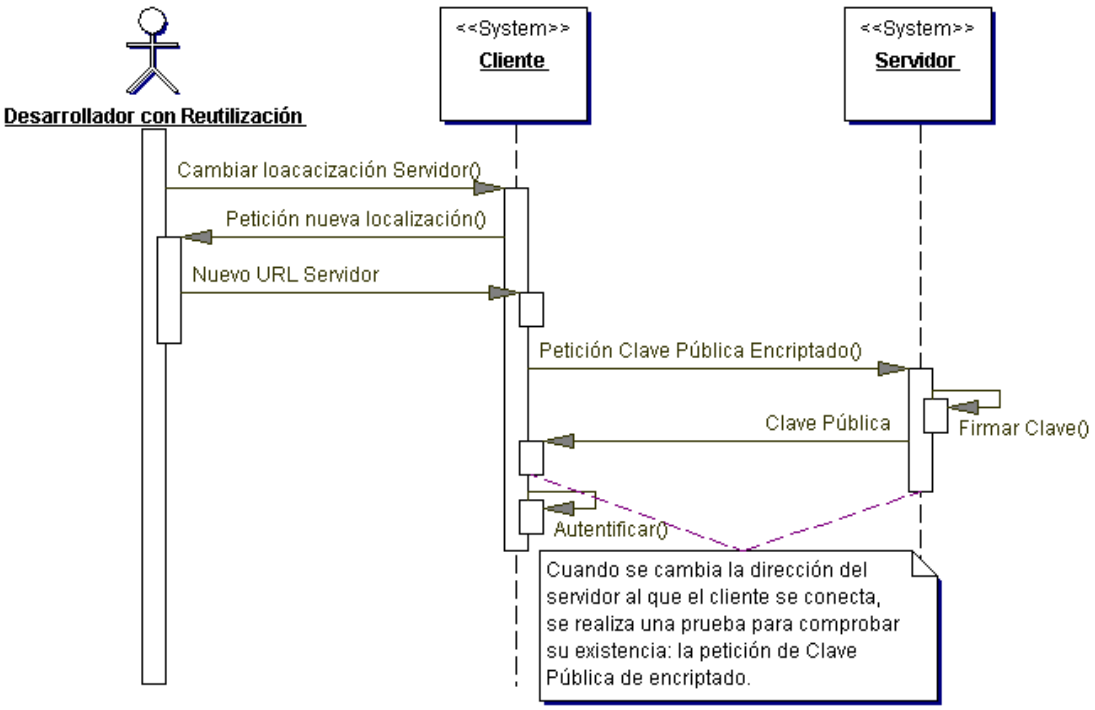

Figura 5.4.- Escenario: Cambiar localización del servidor

5.5.5. Caso de Uso: Cerrar sesión

Con este Caso de Uso el usuario elimina su identificación en el sistema. A partir de entonces el sistema pedirá su identificación al usuario si intenta realizar alguna acción que requiera identificación.

Caso de uso	CS-1.03 Cerrar sesión	
Actores	Desarrollador con reutilización.	
Descripción	Mediante este Caso de Uso el usuario elimina su identificación en el sistema. A partir de entonces el sistema pedirá la identificación al usuario, hasta que se identifique correctamente, si intenta realizar alguna acción que requiera identificación.	
Precondiciones	El usuario se debe haber identificado con éxito en algún momento anterior. Si el usuario no se hubiera identificado anteriormente, el Caso de Uso termina.	
Secuencia normal	Paso	Acción
	1	El actor o usuario pulsa la opción *Cerrar Sesión.*
	2	Solicitud en ventana de diálogo de confirmación para llevar a cabo la acción.
	3	El usuario pulsa el botón OK..
	4	El sistema elimina la identificación de usuario previa.
	5	Se informa al usuario de que su sesión fue cerrada.
Escenarios secundarios	1	No había una sesión abierta, se informa de ello al usuario. Fin del Caso de Uso
	2	El cliente cancela el cierre de sesión (botón CANCEL), se anula el cierre de sesión.
Postcondiciones	1.- Si existía una identificación de usuario en el sistema, ahora no existe. 2.- El sistema solicitará al usuario que se identifique si desea realizar alguna operación que requiera identificación.	

El escenario para este Caso de Uso sería:

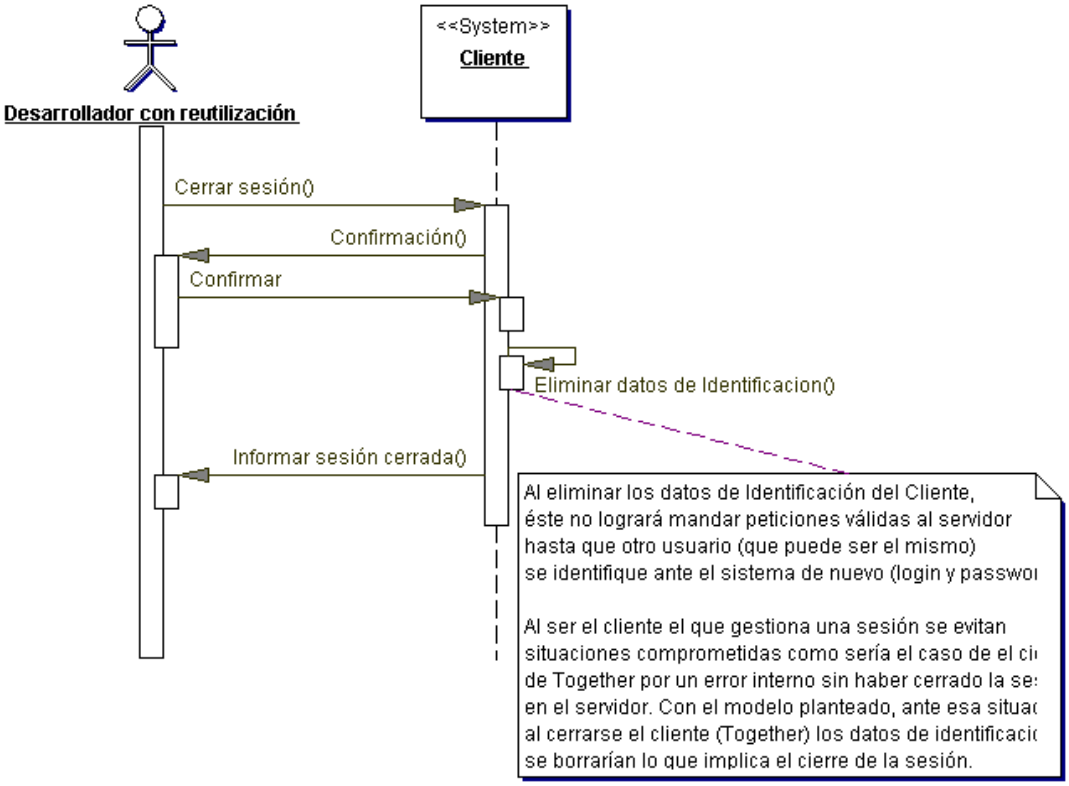

Figura 5.5.- Escenario: Cerrar sesión

5.5.6. Caso de Uso: Validar Usuario

Con este Caso se Uso el Usuario se identifica ante el sistema si no la había hecho antes.

Caso de uso	CS-1.04 Validar usuario	
Actores	No tiene actores; Tiene una relación <<*include*>> con CS–1.07.	
Descripción	El usuario se identifica ante el sistema indicando su login y password. Si la identificación es válida esta quedará en el sistema hasta que el usuario decida cerrar su sesión.	
Precondiciones	1.- Sólo se llamará a este Caso de Uso cuando el usuario la operación principal CS-1.07. 2.- Debe haber un proyecto abierto (la mantiene el Caso de Uso que llama a éste). 3.- El sistema debe poseer la clave pública de encriptado del servidor al que mandará la identificación del usuario.	
Secuencia normal	Paso	Acción

	1	Se comprueba si el usuario se identificó anteriormente. - No se identificó : Ir a 2. - Se identificó con anterioridad: Ir a 5
	2	El usuario no se identificó con anterioridad y se solicita al usuario que se identifique (login y password).
	3	El usuario introduce en el cliente su identificación.
	4	El cliente encripta con la clave pública obtenida del servidor la identificación del usuario.
	5	El cliente envía la identificación del usuario al servidor para su validación.
	6	El servidor recibe la identificación del cliente y la desencripta con su clave privada.
	7	El servidor comprueba en la base de datos del repositorio la identificación del usuario.
	8	El servidor envía al cliente la confirmación de identificación del usuario correcta.
	9	El cliente informa al usuario del resultado de la validación y el Caso de Uso finaliza.
Escenarios secundarios	2	El cliente cancela la petición de datos (botón CANCEL), nos vamos al punto 9 (validación errónea).
	5 6	Ocurre un error en la transmisión vía *Internet,* en ese caso ocurrirá una excepción del sistema y se informará al usuario (validación errónea).
	7	El usuario no se encuentra identificado en la base de datos, se va al punto 9 (validación errónea).
Postcondiciones	Se han comprobado la identificación del usuario, continuando con la operación seleccionada dependiendo la llamada al Caso de Uso.	

El escenario correspondiente a este Caso de Uso es:

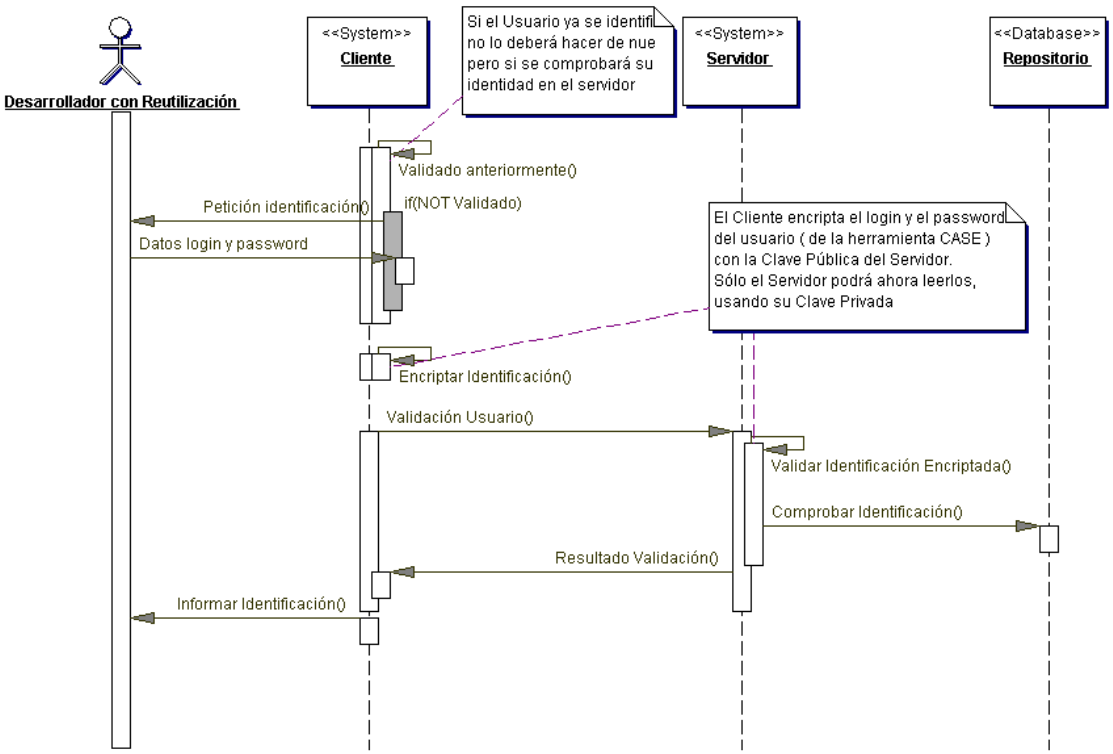

Figura 5.6.- Escenario: Validar Usuario.

5.5.7. Caso de Uso: Buscar Proyecto

Con este Caso de Uso el cliente busca en el repositorio los elementos reutilizables que se adaptan a las necesidades del Desarrollador con Reutilización y que previamente éste le ha indicado al cliente.

Caso de uso	CS-1.05 Buscar Proyecto
Actores	No tiene actores; Tiene una relación <<*include*>> con CS–1.07.
Descripción	El Desarrollador con Reutilización indica al sistema que características han de cumplir los elementos reutilizables que él necesita. El sistema consigue una lista para uso interno de los elementos reutilizables que cumplen dichas condiciones

Precondiciones	1.- Sólo se llamará a este Caso de Uso cuando el usuario la operación principal CS-1.07. 2.- Debe haber un proyecto abierto (la mantiene el Caso de Uso que llama a éste). 3.- El sistema debe poseer la clave pública de encriptado del servidor al que mandará la identificación del usuario. 4.- La identificación del usuario ha debido ser validada en un momento anterior.	
Secuencia normal	Paso	Acción
	1	El cliente solicita al usuario (desarrollador con reutilización) las características o patrones que deben seguir los elementos reutilizables que el usuario desee recuperar.
	2	El usuario introduce las características o patrones seleccionados y pulsa el botón OK.
	3	El cliente añade la identificación del usuario encriptada a su petición al servidor.
	4	El cliente envía el patrón de búsqueda introducido al servidor.
	5	El servidor recibe la identificación del cliente y la desencripta con su clave privada.
	6	El servidor comprueba en la base de datos del repositorio la identificación del usuario.
	7	Tras una verificación de la identidad del usuario correcta, el servidor busca en el repositorio los elementos reutilizables que encajen en el patrón de búsqueda enviado por el usuario.
	8	El repositorio envía al servidor la información de los elementos reutilizables que buscaba.
	9	El servidor envía al cliente la información de los elementos reutilizables buscados.
Escenarios secundarios	2	El cliente cancela la petición de datos (botón CANCEL), se va al punto 9.
	4 9	Ocurre un error en la transmisión vía *Internet,* en ese caso ocurrirá una excepción del sistema y se informará al usuario.

	6	El usuario no se encuentra identificado en la base de datos, se va al punto 9.
Postcondiciones	Se ha obtenido la información lógica relevante de los elementos reutilizables que encajan en el patrón de búsqueda indicado por el usuario.	

El escenario correspondiente a este Caso de Uso es:

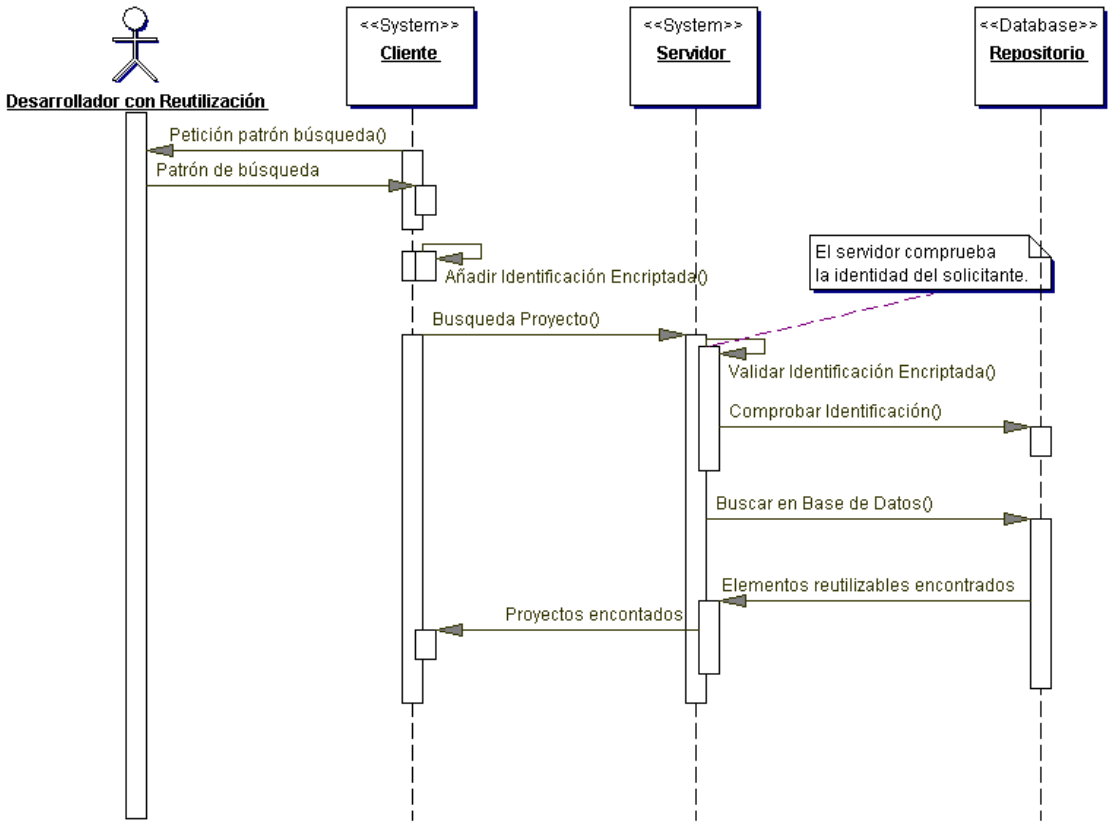

Figura 5.7.- Buscar Proyecto

5.5.8. Caso de Uso: Seleccionar Proyecto a Importar

Con este Caso de Uso el usuario selecciona que elemento reutilizable de los que encajan en el patrón de búsqueda que él indico al sistema con anterioridad.

Caso de uso	CS-1.06 Seleccionar Proyecto a Importar
Actores	No tiene actores; Tiene una relación <<*include*>> con CS–1.07.

Descripción	El Desarrollador con Reutilización indica al sistema que elemento reutilizable, de los que encajan en el patrón de búsqueda que el indico al sistema con anterioridad, desea extraer del repositorio e insertarlo en el proyecto Together en el que se encuentra trabajando. Además el usuario podrá ver detalladamente las características de cada elemento reutilizable listado antes de seleccionarlo definitivamente para su importación.	
Precondiciones	1.- Sólo se llamará a este Caso de Uso cuando el usuario ejecute la operación principal CS-1.07. 2.- Debe haber un proyecto abierto (la mantiene el Caso de Uso que llama a éste). 3.- El sistema debe poseer la clave pública de encriptado del servidor al que mandará la identificación del usuario. 4.- La identificación del usuario ha debido ser validada en un momento anterior. 5.- El sistema dispone de la lista de elementos reutilizables que encajan el en patrón de búsqueda indicado por el usuario.	
Secuencia normal	Paso	Acción
	1	El cliente solicita al usuario (desarrollador con reutilización) la selección de un elemento reutilizable para que se importe en el proyecto Together activo.
	2	El usuario indica al sistema que desea ver los detalles de un elemento reutilizable en concreto.
	3	El cliente muestra al usuario los detalles de la información asociada a cada elemento reutilizable (nombre, fecha de introducción en el repositorio, versión, resumen, etc...)
	4	El cliente selecciona finalmente un elemento reutilizable para su inserción en el proyecto Together en el que se encontrara trabajando.
Escenarios secundarios	2	El usuario cancela la selección (botón CANCEL), se va al punto 9.
Postcondiciones	Se ha obtenido una identificación unívoca de que elemento reutilizable desea el usuario importar a su trabajo.	

El escenario correspondiente e este Caso de Uso sería:

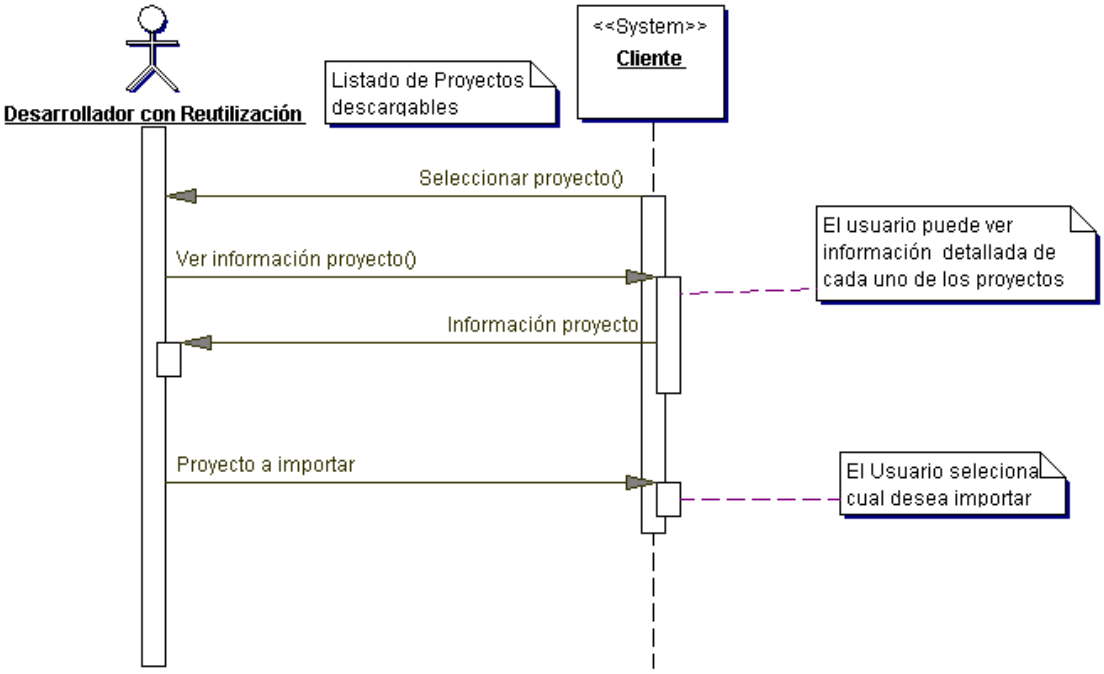

Figura 5.8.- Escenario Seleccionar Proyecto a Importar.

5.5.9 Caso de Uso: Importar Proyecto Together

Con este Caso de Uso se importa un elemento reutilizable (y todos los elementos reutilizables necesarios para su reutilización) desde el repositorio al proyecto Together en el que el usuario estuviera trabajando.

Caso de uso	CS-1.07 Importar Proyecto Together	
Actores	Desarrollador con Reutilización	
Descripción	El Desarrollador con Reutilización indica al sistema que desea incorporar un elemento reutilizable (y todos los elementos reutilizables necesarios para su reutilización) desde el repositorio GIRO al proyecto Together en el que el usuario estuviera trabajando. El usuario indicará al sistema un patrón de búsqueda del elemento reutilizable que el desea y seleccionará un solo elemento reutilizable para su extracción del repositorio e inserción en el proyecto en el que el usuario se encontrara trabajando.	
Precondiciones	Ninguna	
Secuencia normal	Paso	Acción
	1	El actor o usuario pulsa la opción *ImportarMecano* o

		Importar Asset.
	2	El sistema comprueba que hay en la herramienta Together un proyecto abierto sobre el que importar elementos reutilizables.
	3	El cliente solicita al servidor una copia actualizada de la clave pública de encriptado.
	4	El servidor envía al cliente una clave pública de encriptado actualizada y éste último valida que su autenticidad.
	5	VER Caso de Uso: Validar Usuario
	6	VER Caso de Uso: Buscar Proyecto
	7	VER Caso de Uso: Seleccionar Proyecto a Importar
	8	El cliente añade la identificación encriptada del usuario a su próxima petición al servidor.
	9	El cliente envía al servidor una petición de descarga de un elemento reutilizable
	10	El servidor comprueba la validez de la identificación encriptada enviada por el cliente
	11	El servidor contrasta la identificación del cliente con su identificación en el repositorio.
	12	Con una identificación correcta, se extrae del repositorio el elemento reutilizable a descargar.
	13	El servidor comprueba las dependencias existentes en el repositorio de este elemento a extraer.
	14	Mientras existan dependencias que resolver se extraerán
	14.a	Se extrae el asset del que depende el elemento reutilizable o los assets que a su vez dependen de él.
	14.b	Se comprueban las dependencias existentes con el asset recién extraído.
	15	Resueltas todas las dependencias, se empaquetan todos los elementos reutilizables extraídos.
	16	El servidor envía el proyecto resultante al cliente.
	17	El cliente construye un proyecto Together con los elementos reutilizables extraídos del repositorio, uniéndolo al proyecto Together en el que el usuario se encontraba

		trabajando.
	18	El sistema informa al usuario durante todo el proceso de construcción.
	19	Terminada la operación, el sistema informa al usuario del resultado de la importación del proyecto. El Caso de Uso termina.
Escenarios secundarios	5 6 7	El usuario cancela la operación (botón CANCEL) en cualquier instante en que se solicite su colaboración, se va al punto 19.
	3 4 5 6 9 16	Ocurre un error en la transmisión vía *Internet,* en ese caso ocurrirá una excepción del sistema y se informará al usuario.
	5 6 11	El usuario no se encuentra identificado en la base de datos, se va al punto 19.
	12	El repositorio (la base de datos del repositorio) es inconsistente y no se puede encontrar la representación física del elemento reutilizable a descargar. Se cancela la importación y se informa al usuario
	14.a	El repositorio (la base de datos del repositorio) es inconsistente y no se puede encontrar un asset del que depende (directa o indirectamente) el elemento reutilizable a descargar. Ese asset se ignora durante toda la operación no incluyéndole en el proyecto a enviar al cliente y tampoco se comprueban sus dependencias, se pasa la siguiente dependencia.
Postcondiciones	Se ha insertado en el proyecto actual de trabajo de Together un elemento reutilizable y todos aquellos assets necesarios para su correcta reutilización.	

El escenario correspondiente a este caso de uso es:

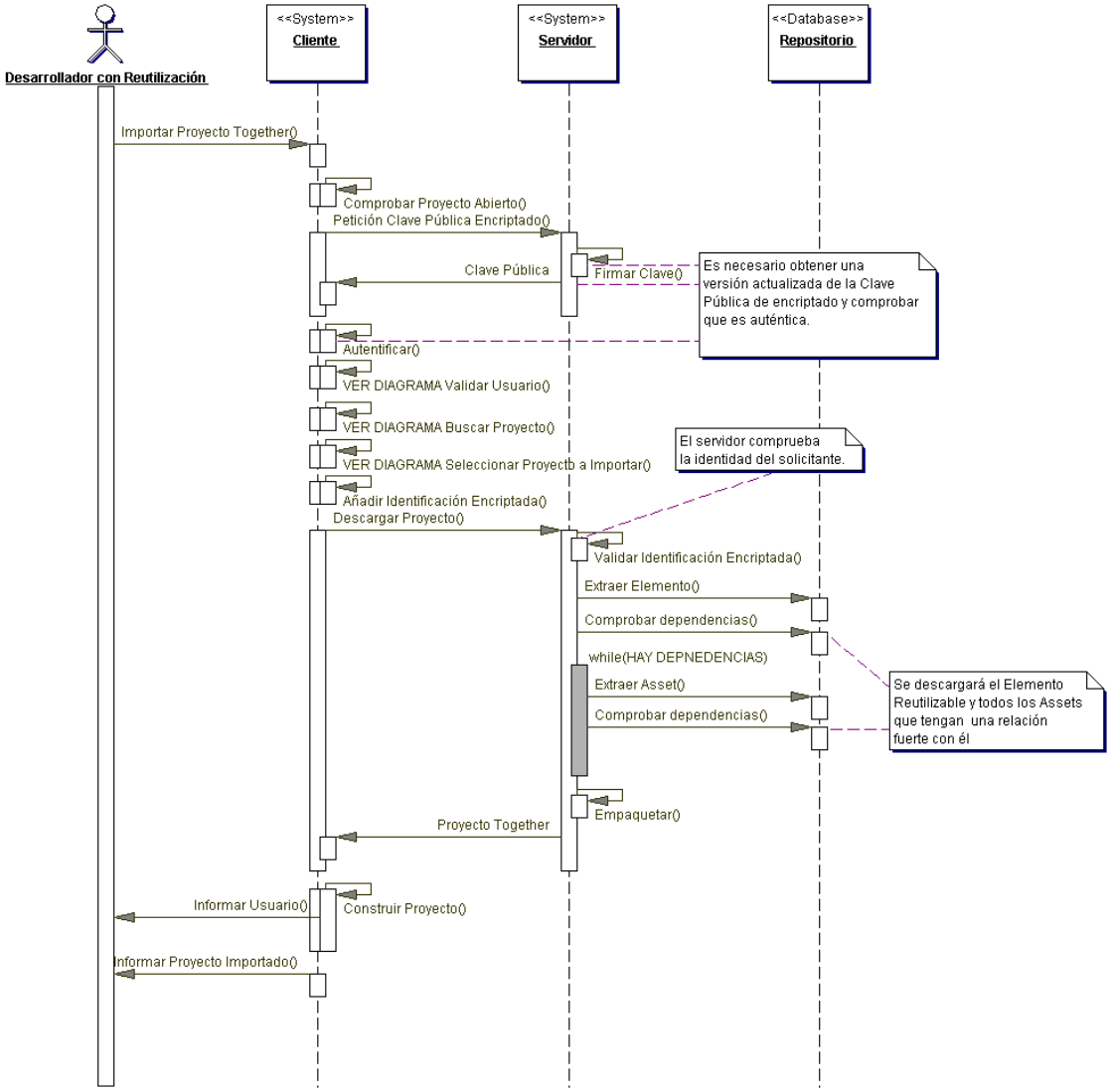

Figura 5.9.- Escenario Importar Proyecto Together.

5.5.10. Caso de Uso: Importar Asset

Con este Caso de Uso se importa un Asset (y todos los assets necesarios para su reutilización) desde el repositorio al proyecto Together en el que el usuario estuviera trabajando.

Caso de uso	CS-1.08 Importar Asset
Actores	No tiene actores, tiene una relación <<*extend*>> con el CS-1.07 en el Extension Point **[a]**

Descripción	El Desarrollador con Reutilización indica al sistema que desea incorporar un asset (y todos los assets necesarios para su reutilización) desde el repositorio GIRO al proyecto Together en el que el usuario estuviera trabajando. El usuario indicará al sistema un patrón de búsqueda del asset que el desea y seleccionará un solo asset para su extracción del repositorio e inserción en el proyecto en el que el usuario se encontrara trabajando.	
Precondiciones	Ninguna	
Secuencia normal	Paso	Acción
	1	El actor o usuario pulsa la opción *Importar Asset.*
	2	El sistema comprueba que hay en la herramienta Together un proyecto abierto sobre el que importar el asset
	3	El cliente solicita al servidor una copia actualizada de la clave pública de encriptado.
	4	El servidor envía al cliente una clave pública de encriptado actualizada y éste último valida su autenticidad.
	5	VER Caso de Uso: Validar Usuario
	6	VER Caso de Uso: Buscar Proyecto
	7	VER Caso de Uso: Seleccionar Proyecto a Importar
	8	El cliente añade la identificación encriptada del usuario a su próxima petición al servidor.
	9	El cliente envía al servidor una petición de descarga de un asset.
	10	El servidor comprueba la validez de la identificación encriptada enviada por el cliente
	11	El servidor contrasta la identificación del cliente con su identificación en el repositorio.
	12	Con una identificación correcta, se extrae del repositorio el asset a descargar.
	13	El servidor comprueba las dependencias existentes en el repositorio de este asset a extraer.
	14	Mientras existan dependencias que resolver se extraerán.
	14.a	Se extrae el asset del que depende el asset principal

		extraído o los assets que a su vez dependen de él.
	14.b	Se comprueban las dependencias existentes con el asset recién extraído.
	15	Resueltas todas las dependencias, se empaquetan todos los assets extraídos.
	16	El servidor envía el proyecto resultante al cliente.
	17	El cliente construye un proyecto Together con los assets extraídos del repositorio, uniéndolo al proyecto Together en el que el usuario se encontraba trabajando.
	18	El sistema informa al usuario durante todo el proceso de construcción.
	19	Terminada la operación, el sistema informa al usuario del resultado de la importación del proyecto. El Caso de Uso termina.
Escenarios secundarios	5 6 7	El usuario cancela la operación (botón CANCEL) en cualquier instante en que se solicite su colaboración, se va al punto 19.
	3 4 5 6 9 16	Ocurre un error en la transmisión vía *Internet,* en ese caso ocurrirá una excepción del sistema y se informará al usuario.
	5 6 11	El usuario no se encuentra identificado en la base de datos, se va al punto 19.
	12	El repositorio (la base de datos del repositorio) es inconsistente y no se puede encontrar la representación física del asset a descargar. Se cancela la importación y se informa al usuario.

	14.a	El repositorio (la base de datos del repositorio) es inconsistente y no se puede encontrar un asset del que depende (directa o indirectamente) el asset principal a descargar. Ese asset se ignora durante toda la operación no incluyéndole en el proyecto a enviar al cliente y tampoco se comprueban sus dependencias, se pasa la siguiente dependencia.
Postcondiciones	Se ha insertado en el proyecto actual de trabajo de Together un asset y todos aquellos assets necesarios para su correcta reutilización.	

El escenario correspondiente a este Caso de Uso sería:

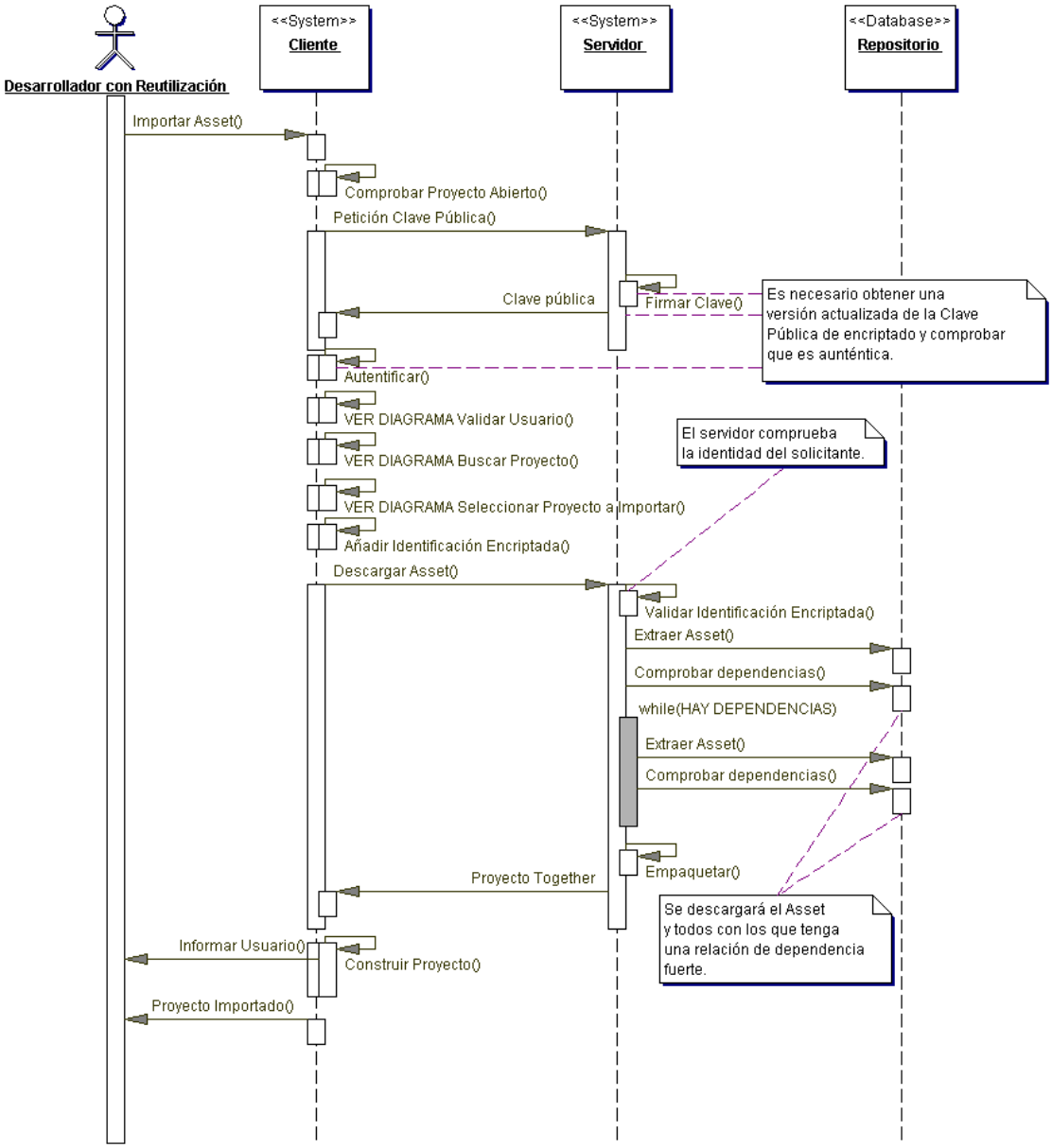

Figura 5.10.- Escenario Importar Asset

5.5.11. Caso de Uso: Importar Mecano

Con este Caso de Uso se importa un Mecano (con todos los assets que lo componen y todos los assets necesarios para la reutilización de los anteriores) desde el repositorio al proyecto Together en el que el usuario estuviera trabajando.

Caso de uso	CS-1.09 Importar Mecano	
Actores	No tiene actores, tiene una relación <<*extend*>> con el CS-1.07 en el Extension Point **[b]**	
Descripción	El Desarrollador con Reutilización indica al sistema que desea incorporar un Mecano (con todos los assets que lo componen y todos los assets necesarios para la reutilización de los anteriores) desde el repositorio GIRO al proyecto Together en el que el usuario estuviera trabajando. El usuario indicará al sistema un patrón de búsqueda del Mecano que el desea y seleccionará un solo Mecano para su extracción del repositorio e inserción en el proyecto en el que el usuario se encontrara trabajando.	
Precondiciones	Ninguna	
Secuencia normal	Paso	Acción
	1	El actor o usuario pulsa la opción *Importar Mecano.*
	2	El sistema comprueba que hay en la herramienta Together un proyecto abierto sobre el que importar el Mecano
	3	El cliente solicita al servidor una copia actualizada de la clave pública de encriptado.
	4	El servidor envía al cliente una clave pública de encriptado actualizada y éste último valida su autenticidad.
	5	VER Caso de Uso: Validar Usuario
	6	VER Caso de Uso: Buscar Proyecto
	7	VER Caso de Uso: Seleccionar Proyecto a Importar
	8	El cliente añade la identificación encriptada del usuario a su próxima petición al servidor.
	9	El cliente envía al servidor una petición de descarga de un Mecano.
	10	El servidor comprueba la validez de la identificación encriptada enviada por el cliente.

	11	El servidor contrasta la identificación del cliente con su identificación en el repositorio.
	12	Con una identificación correcta, se listan todos los assets del repositorio que están ligados a ese Mecano (un asset puede pertenecer a más de un Mecano)
	13	El servidor extrae del repositorio todos los assets listados anteriormente.
	14	El servidor comprueba las dependencias existentes en el repositorio de los assets a extraídos.
	15	Mientras existan dependencias que resolver se extraerán.
	15.a	Se extrae el asset del que depende los assets principales extraídos o los assets que a su vez dependen de ellos.
	15.b	Se comprueban las dependencias existentes con el asset recién extraído.
	16	Resueltas todas las dependencias, se empaquetan todos los assets extraídos.
	17	El servidor envía el proyecto resultante al cliente.
	18	El cliente construye un proyecto Together con los assets extraídos del repositorio, uniéndolo al proyecto Together en el que el usuario se encontraba trabajando.
	19	El sistema informa al usuario durante todo el proceso de construcción.
	20	Terminada la operación, el sistema informa al usuario del resultado de la importación del proyecto. El Caso de Uso termina.
Escenarios secundarios	5 6 7	El usuario cancela la operación (botón CANCEL) en cualquier instante en que se solicite su colaboración, se va al punto 10.
	3 4 5 6 9 17	Ocurre un error en la transmisión vía *Internet,* en ese caso ocurrirá una excepción del sistema y se informará al usuario.

	5 6 11	El usuario no se encuentra identificado en la base de datos, se va al punto 20.
	13	El repositorio (la base de datos del repositorio) es inconsistente y no se puede encontrar ninguna representación física de los assets a descargar. Se cancela la importación y se informa al usuario. Si la inconsistencia fuera puntual (sólo afectara a un asset pero existieran otros consistentes, este asset inconsistente y sus dependencias se ignoran.
	15.a	El repositorio (la base de datos del repositorio) es inconsistente y no se puede encontrar un asset del que dependen (directa o indirectamente) los assets principales a descargar. Ese asset se ignora durante toda la operación no incluyéndole en el proyecto a enviar al cliente y tampoco se comprueban sus dependencias, se pasa la siguiente dependencia.
Postcondiciones	Se ha insertado en el proyecto actual de trabajo de Together un Mecano y todos aquellos assets que lo componían o eran necesarios para su correcta reutilización.	

El escenario correspondiente a esta Caso de Uso es:

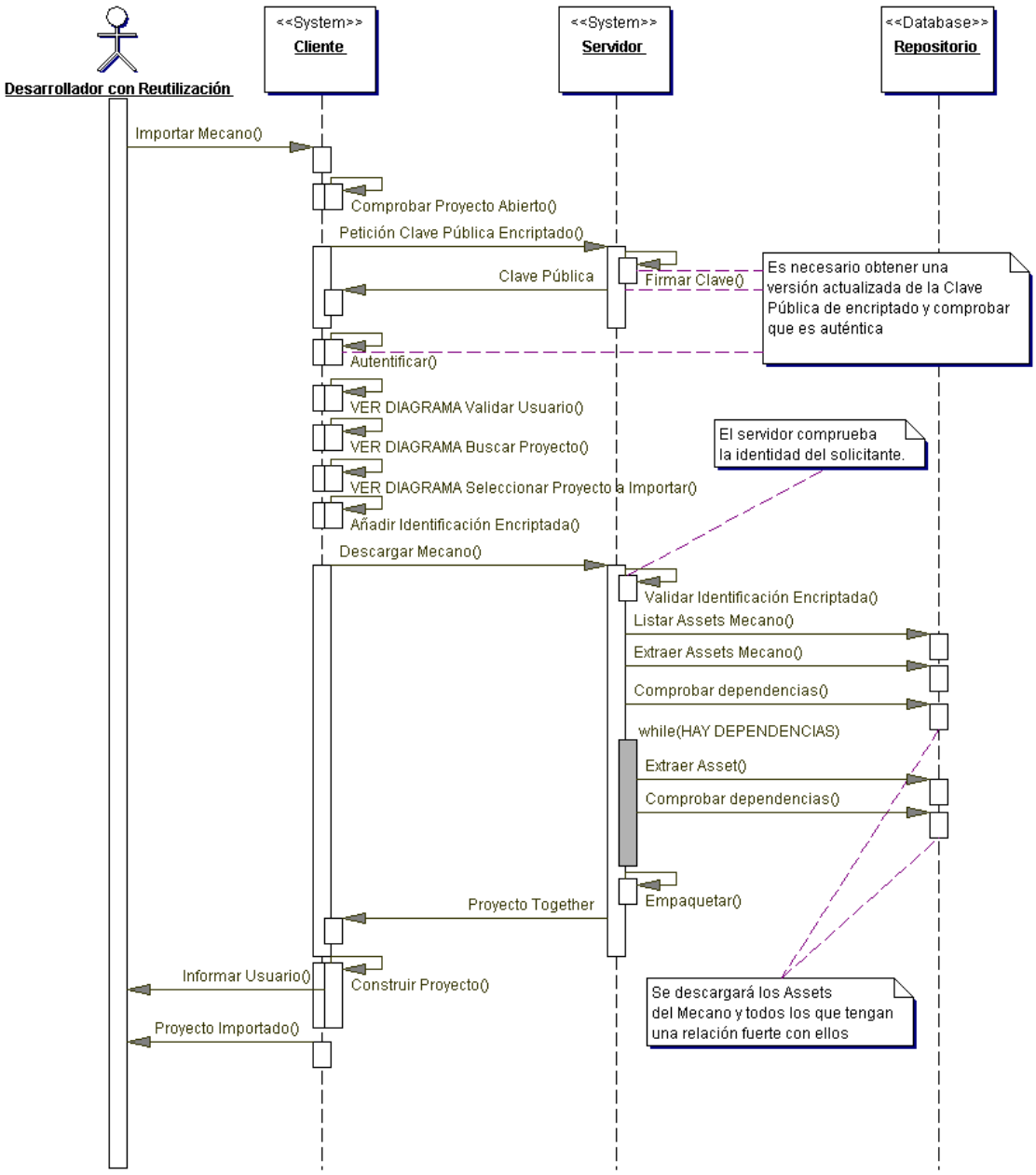

Figura 5.11.- Escenario Importar Mecano

6. DOCUMENTO DE DISEÑO

6.1. Introducción

El diseño es una etapa estratégica en el ciclo de desarrollo de un sistema software con el que se pretende encontrar soluciones técnicas para implementar y construir un sistema software que se adapte a unos requisitos dados.

Partiendo de los requisitos del sistema expuestos en el apartado anterior los pasos a seguir para detallar el diseño del sistema serán:

- Dar una vista simplificada de la arquitectura del sistema a través de diagramas de paquetes.
- Mostrar los diagramas de Clases del sistema.
- Determinar los diagramas de Secuencia.
- Determinar los diagramas de Estados
- Realización del diccionario de clases.

Los tres puntos más destacables de los requisitos y que determinarán claramente la arquitectura del sistema serán:

- El sistema estará basado en una arquitectura cliente-servidor
- El sistema debe funcionar bajo Together (punto de vista del cliente), herramienta totalmente escrita en Java y que sólo admite nuevas funcionalidades en base a módulos.
- Seguridad y privacidad de la identificación de los usuarios del sistema.
- Robustez, el sistema deberá incorporar una gestión completa de errores, que se llevará a la práctica basándose en excepciones (tanto las de los propios lenguajes de programación como las definidas en esta etapa de desarrollo).

6.2. Arquitectura del Sistema

La arquitectura del sistema queda definida por su división en subsistemas que son los componentes que engloba aspectos comunes del sistema (semejante funcionalidad o ubicación física).

La arquitectura puede dividirse en tres grandes subsistemas (cliente, servidor y comunicaciones) pero el subsistema cliente puede dividirse a su vez en otros que se encargan de funcionalidades específicas.

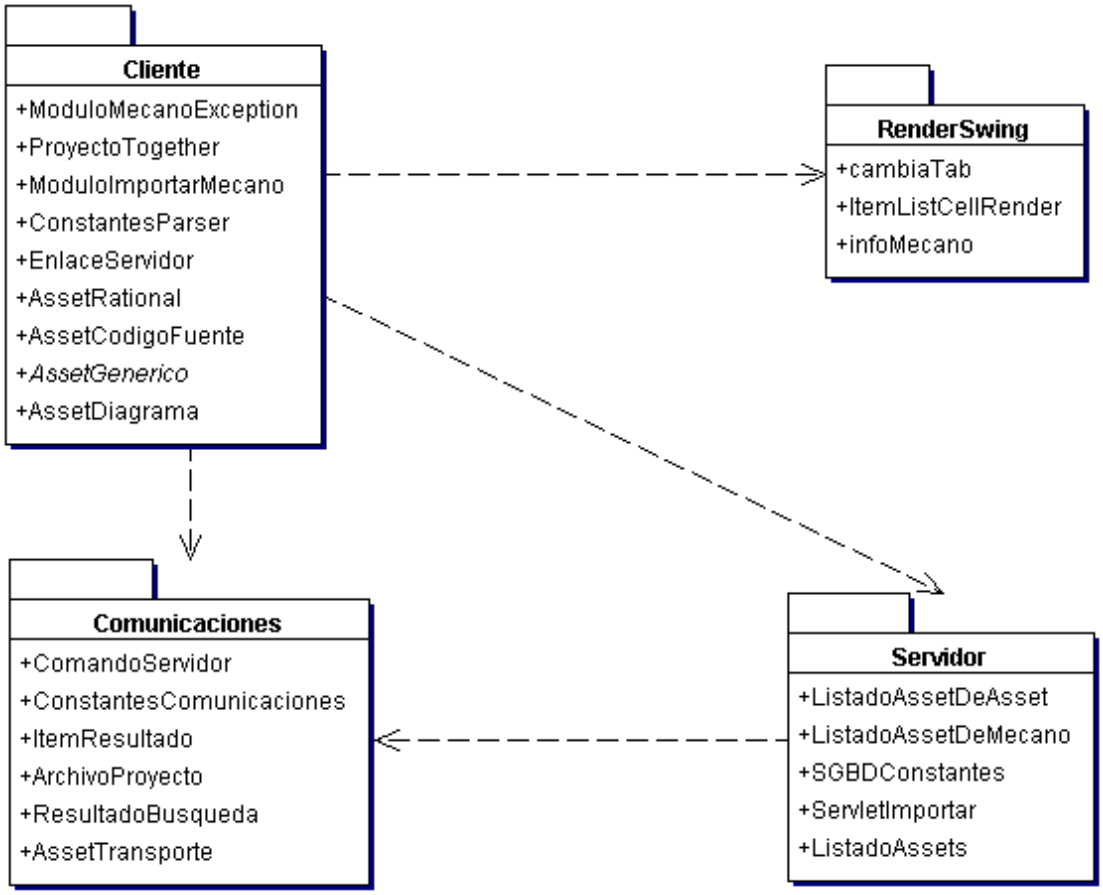

Figura 15.- Arquitectura del sistema.

Basándose en los subsistemas que se observan en la figura anterior, se pasará a una descripción general de cada subsistema, presentados en modo jerárquico para su mejor comprensión:

- Subsistema Cliente:

i) Paquete Cliente:

Se encarga de la funcionalidad del sistema del lado del cliente, y sobre él recaen las operaciones principales que puede llevar a cabo el usuario:

1. Importar Mecano, es decir, insertar un Mecano del repositorio en el proyecto activo de la herramienta Together.
2. Importar Asset, es decir, insertar un Asset del repositorio (y todos los necesarios para su reutilización) en el proyecto activo de la herramienta Together.
3. Modificar la dirección del servidor del repositorio al que el cliente dirige sus peticiones.
4. Cerrar la sesión de usuario para que su identificación se borre del sistema cliente y poder identificarse como un nuevo usuario sin cerrar la herramienta Together.

Del inicio de cada una de estas funcionalidades se encargará la clase ModuloImportarMecano que quedará detallada para su lectura en el diccionario de clases.

ii)Paquete RenderSwing:

Se encarga de una parte específica de la gestión de eventos para el subsistema cliente dependiente de la herramienta Together y el lenguaje de programación que se va emplear. La justificación de su existencia es su labor específica para hacer mas atractiva la interfaz del sistema al usuario.

- Subsistema Servidor:

Este subsistema englobado en el paquete servidor se encargará las labores de comunicación entre el repositorio y el cliente orientado a desarrollo con reutilización.
Sus labores se pueden desglosar en:

1. La búsqueda en el repositorio de elementos reutilizables (tanto assets como mecanos) que concuerden con un patrón de búsqueda indicado desde el cliente.
2. La extracción de los assets requeridos del repositorio por el cliente (y los assets adicionales necesarios par su reutilización) para que le sean enviados (esto incluye tanto la petición de extracción de assets como la de mecanos).
3. Se encargará de la identificación de los usuarios del subsistema cliente para garantizar la seguridad del sistema y la privacidad de los datos de identificación de los usuarios.

- Subsistema Comunicaciones:

Este subsistema se encargará de dar soporte a los datos que intercambien cliente y servidor, así como para centralizar el acceso a valores comunes para ambos sistemas.
Sus labores serán:

1. Mantener una estructura uniforme para el intercambio de órdenes entre cliente y servidor.
2. Mantener una estructura para almacenar valores constates y de consulta común para cliente y servidor.
3. Mantener un estructura para el acceso uniforme a los datos enviados desde el servidor al cliente, tanto para las consultas como para el envío de los elementos reutilizables.

6.3 Diagramas de Clases

Un diagrama de clases es un diagrama que muestra un conjunto de interfaces, colaboraciones y sus relaciones. Gráficamente, un diagrama de clases es una colección de nodos y arcos. Lo que principalmente distingue a este tipo de diagramas es su contenido. Normalmente contienen los siguientes elementos:

- clases
- interfaces
- colaboraciones
- relaciones de dependencia, generalización y asociación

A continuación se muestran los diagramas de clases del sistema:

Cliente:

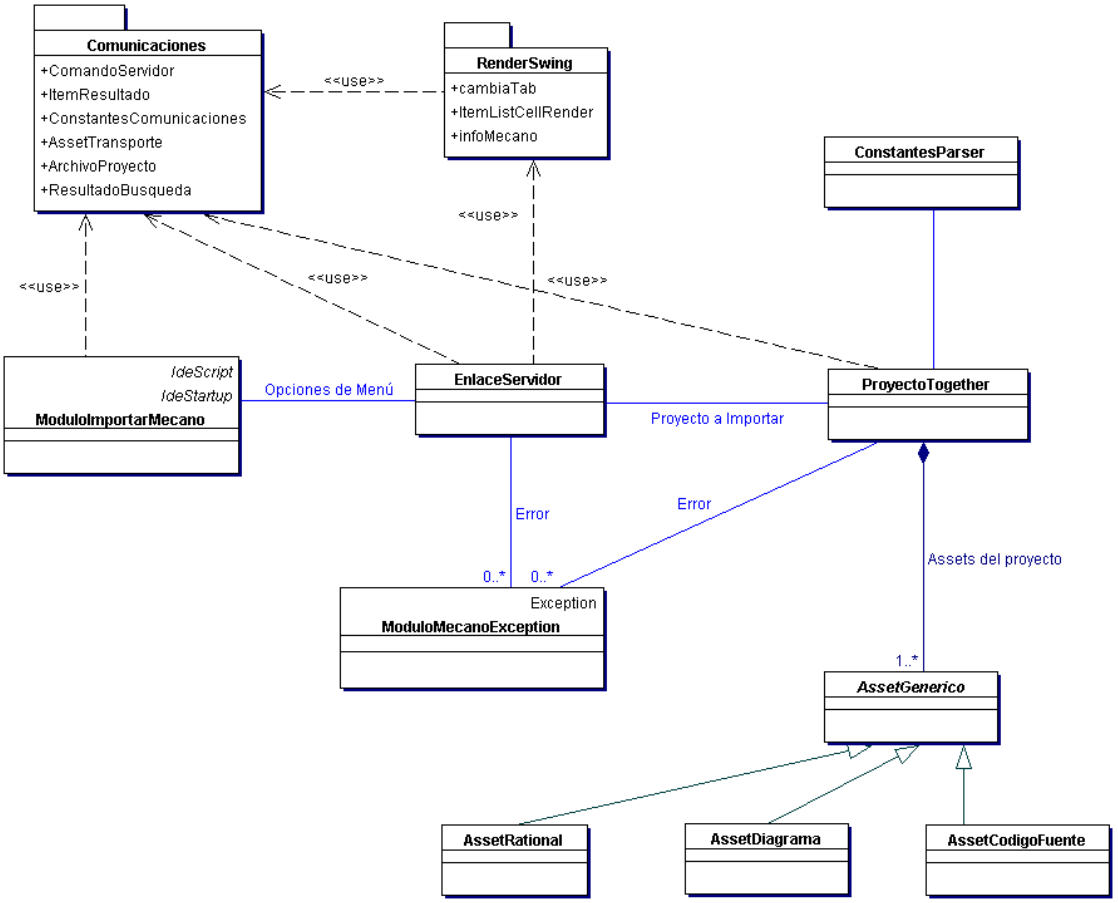

Figura 6.1.- Diagrama de Clases del Cliente

Dentro del cliente los diagramas de clases de los sub-subsistemas cliente serían:

RenderSwing:

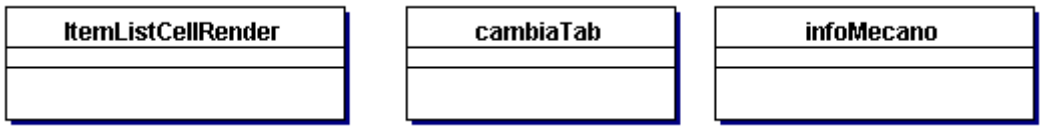

Figura 6.2.- Diagrama de Clases del paquete RenderSwing

Servidor:

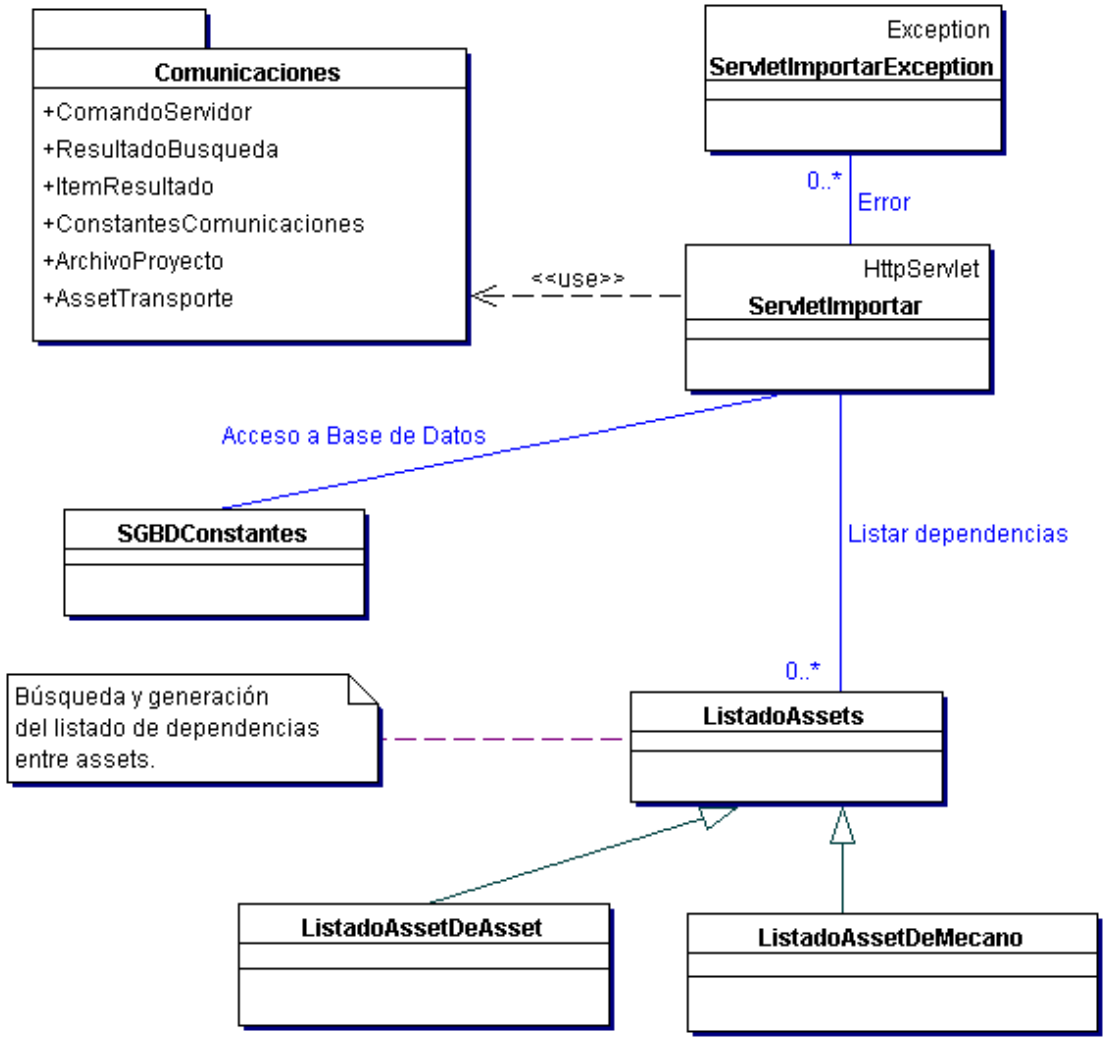

Figura 6.3- Diagrama de Clases del Servidor

Comunicaciones:

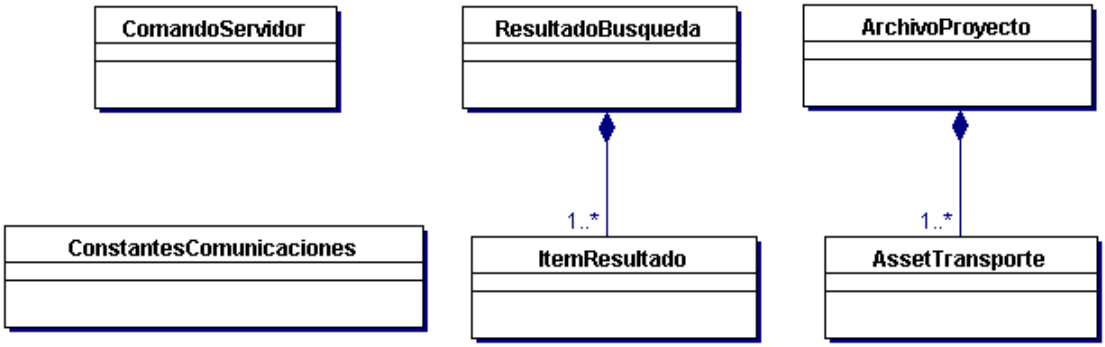

Figura 6.4.- Diagrama de Clases de Comunicaciones

6.4. Diagramas de Secuencia

Un diagrama de secuencia representa una interacción entre objetos insistiendo en la cronología de los envíos de mensajes, con ellos entenderemos la secuencia de eventos que se van produciendo en el sistema.

Con estos diagramas de detallarán las acciones que a nivel de diseño realiza esta aplicación.

Se han dividido estos diagramas en función de donde se realice la secuencia, en el cliente o en el servidor, pero sólo para secuencias muy complejas que impliquen un diagrama muy grande.

6.4.1.1. Cargar módulo

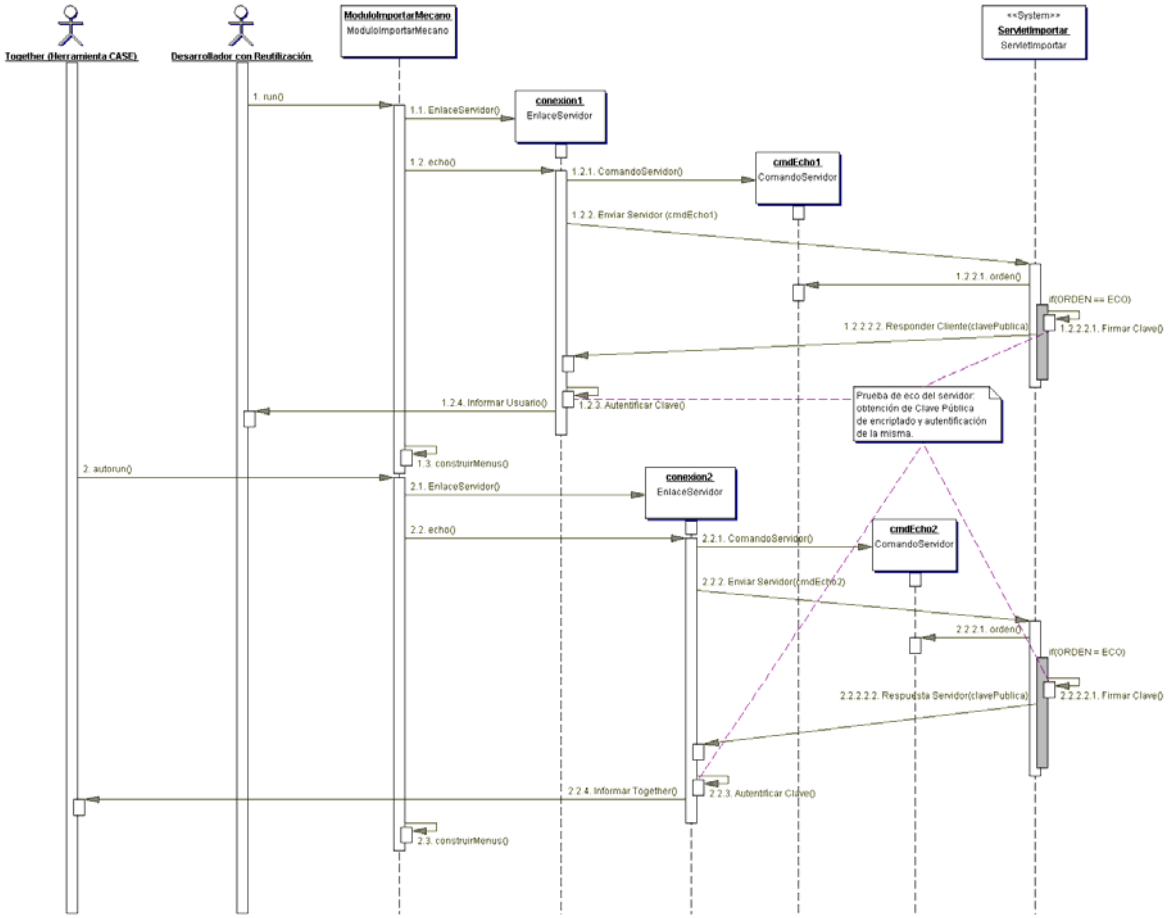

Figura 6.5 .- Secuencia Cargar Módulo

6.4.1.2. Cambiar localización del servidor.

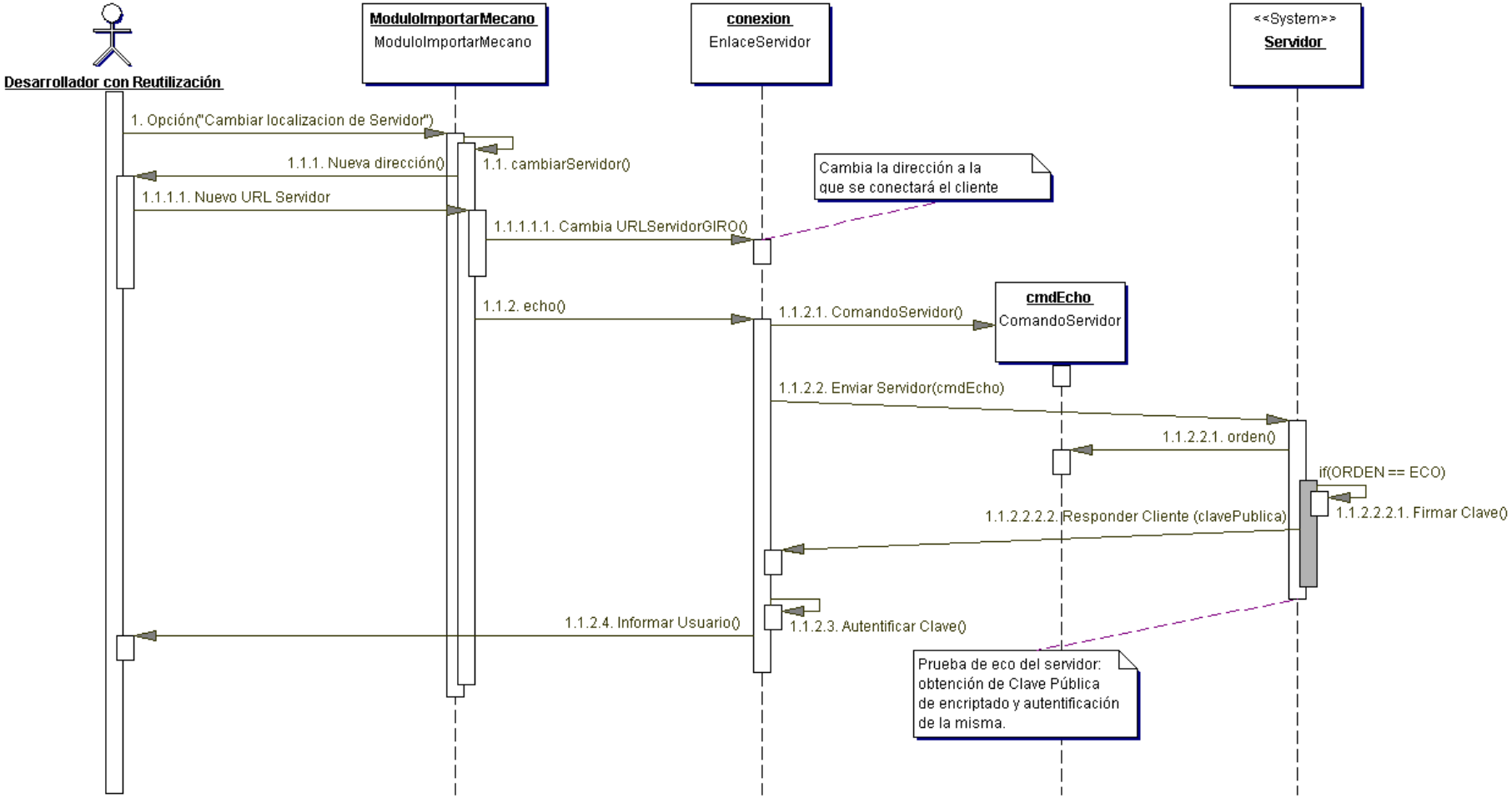

Figura 6.6. Secuencia Cambiar localización del servidor

6.4.1.3. Cerrar Sesión.

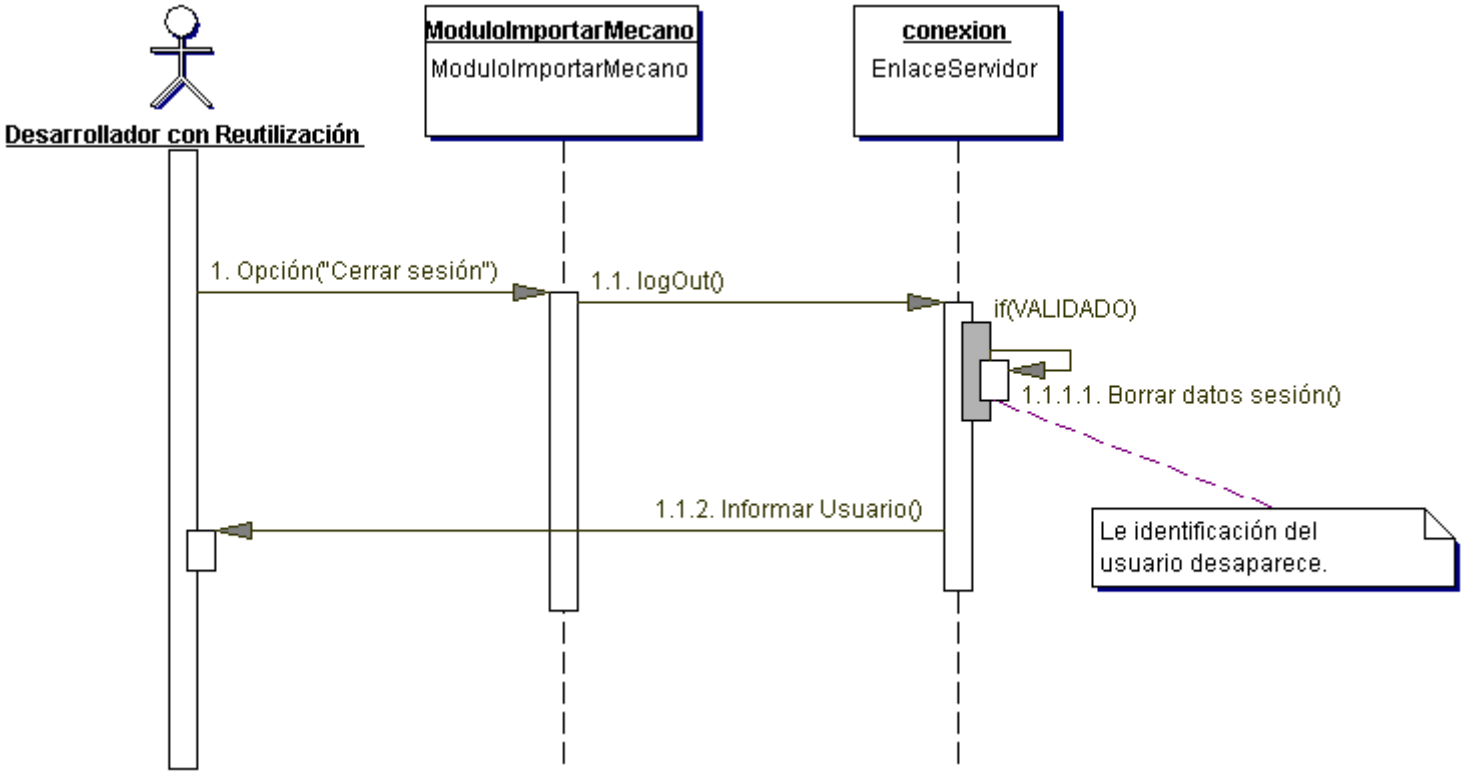

Figura 6.7. Secuencia Cerrar Sesión.

6.4.1.4. Validar Usuario

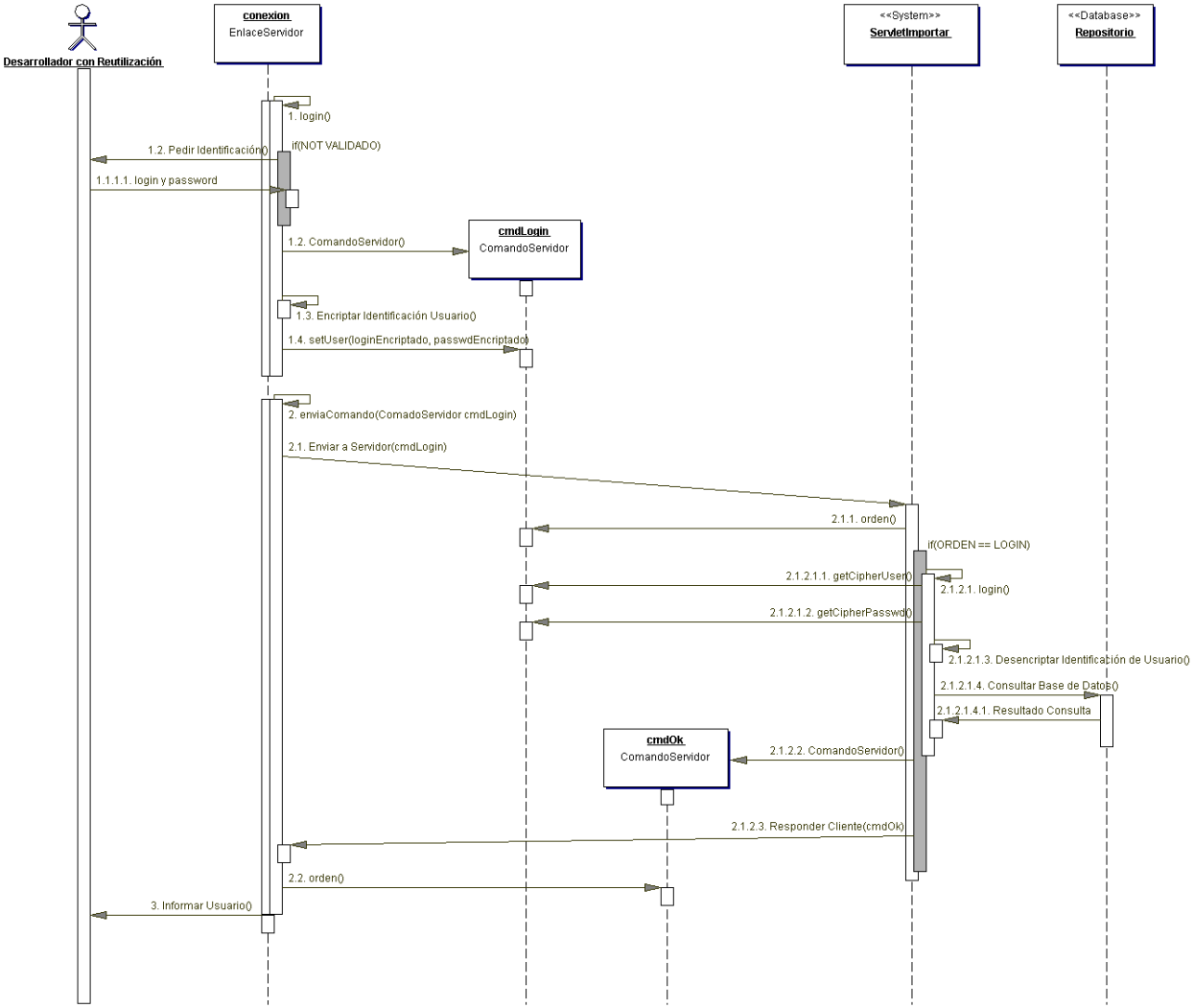

Figura 6.8. Secuencia Validar Usuario.

6.4.1.5. Buscar Proyecto

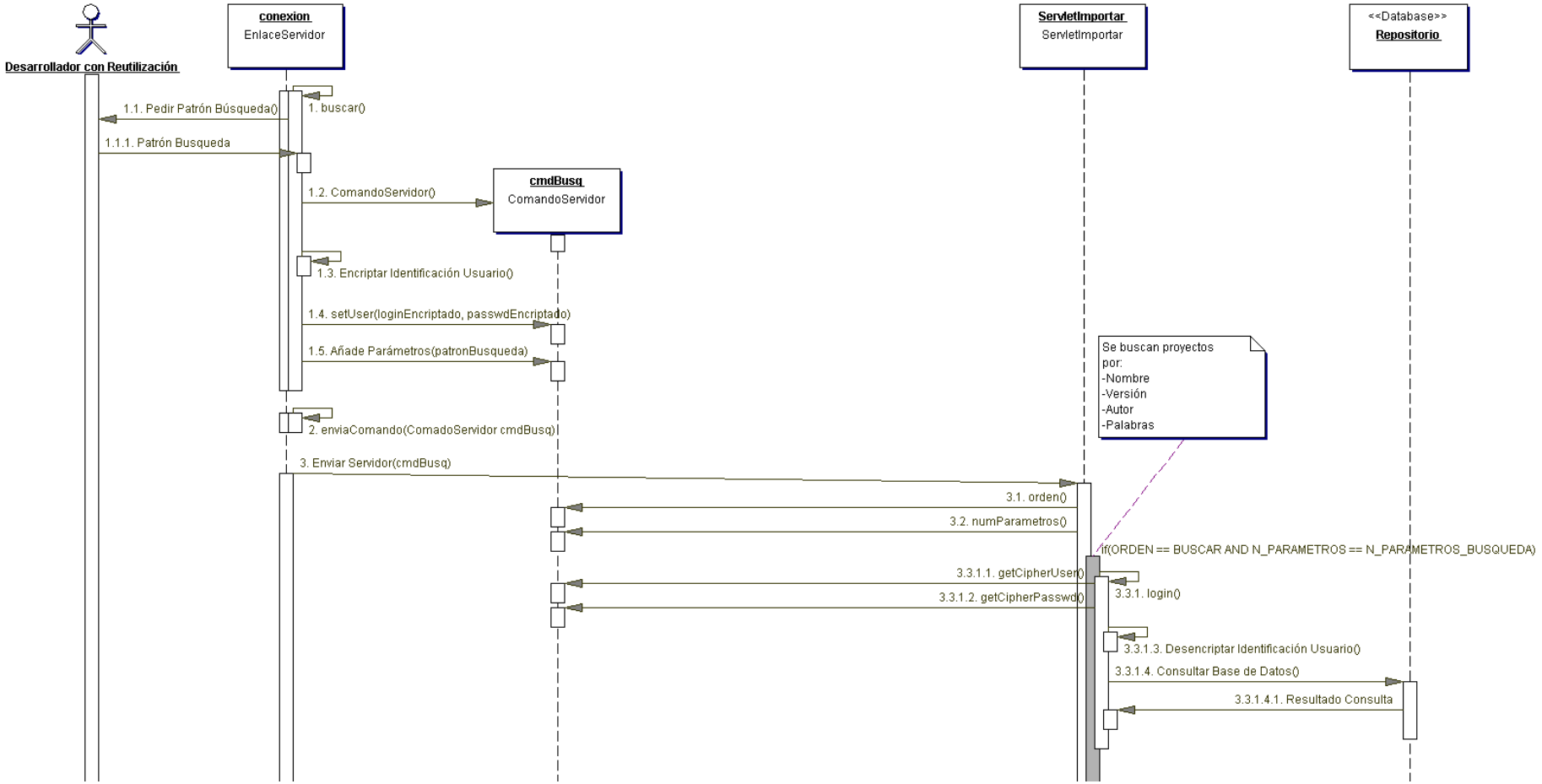

Figura 6.9 Secuencia Buscar Proyecto 1/2

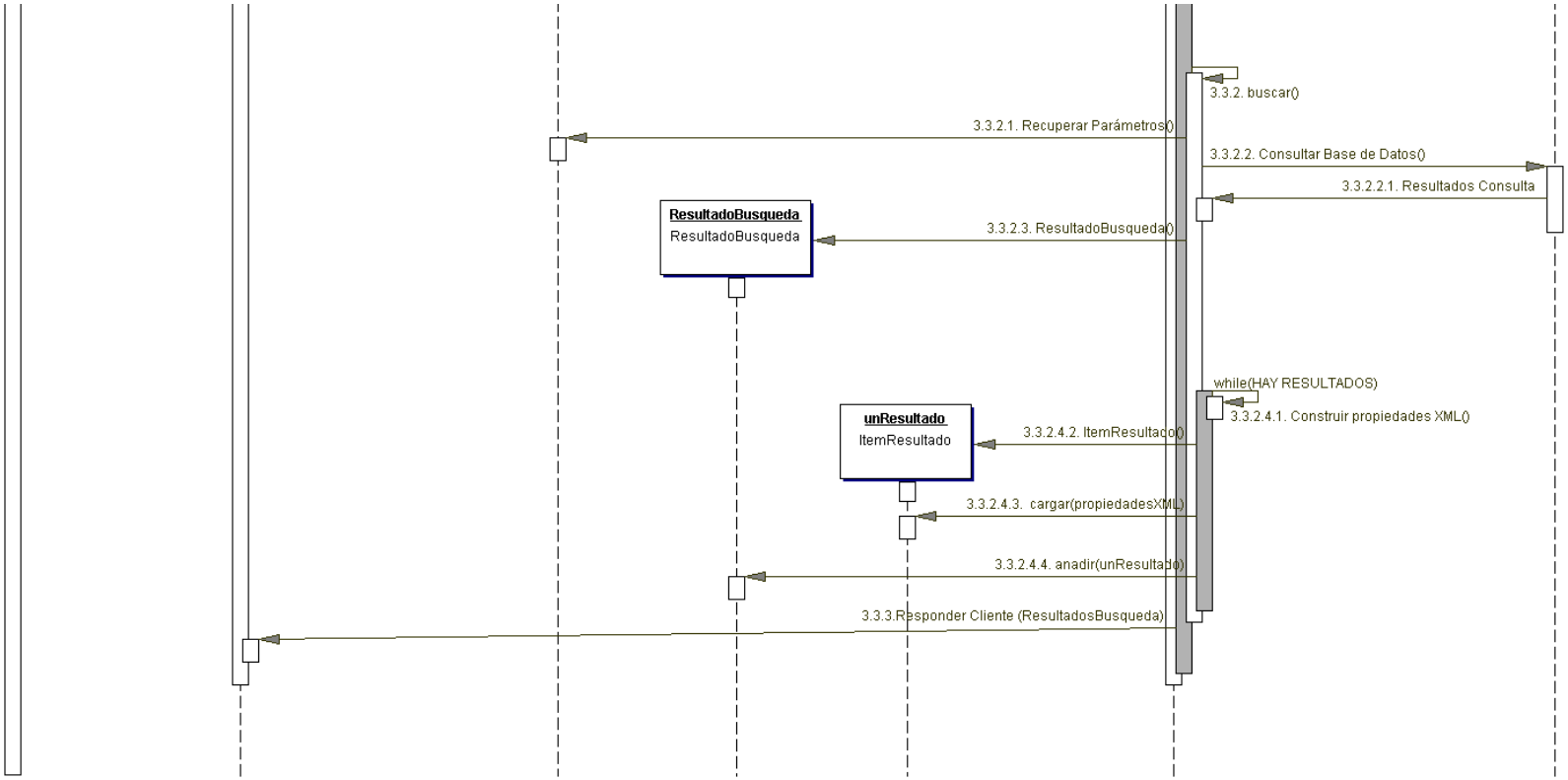

Figura 6.9. Secuencia Buscar Proyecto 2/2

6.4.1.6. Seleccionar Proyecto a Importar

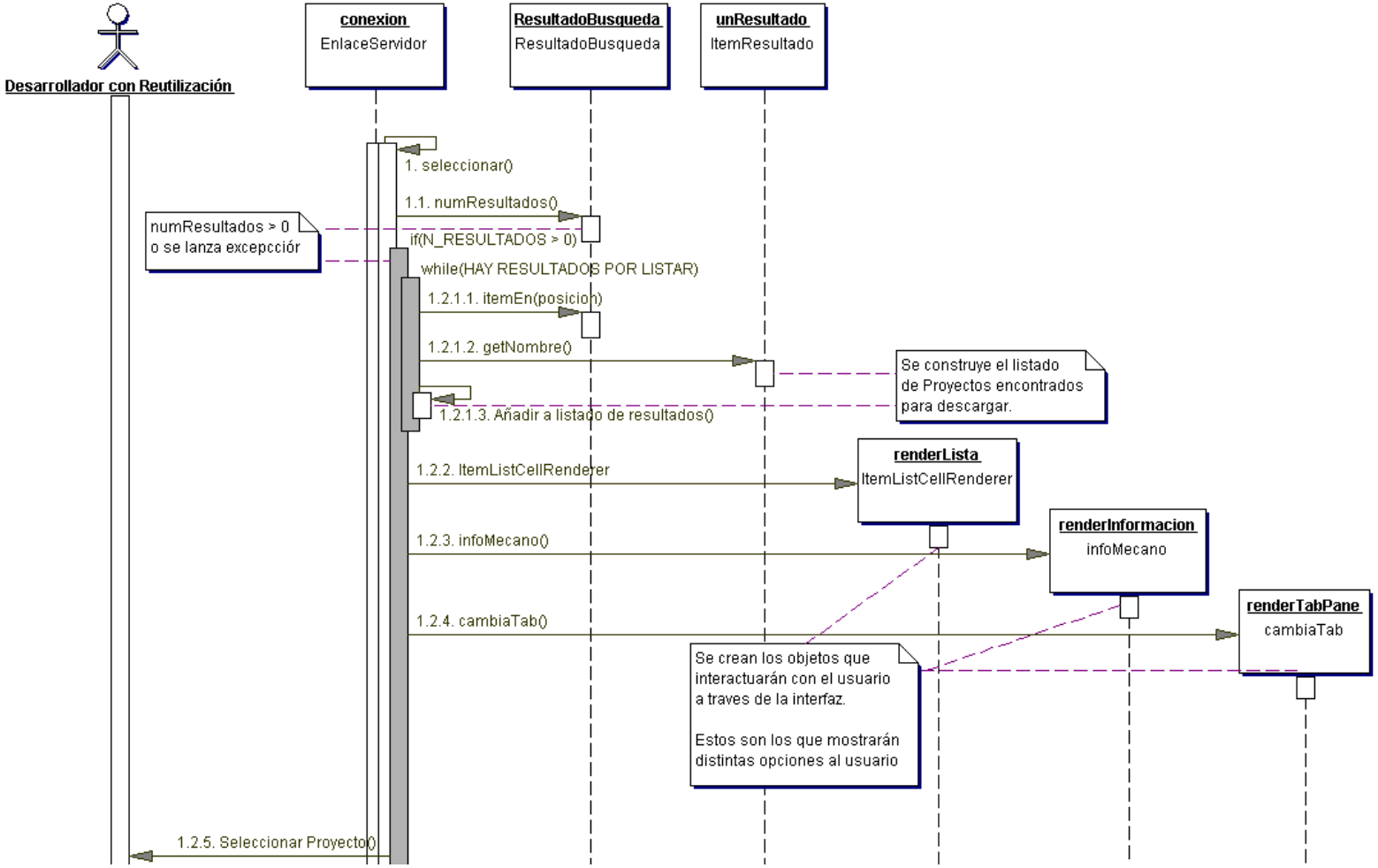

Figura 6.10. Secuencia Seleccionar Proyecto a Importar 1/2

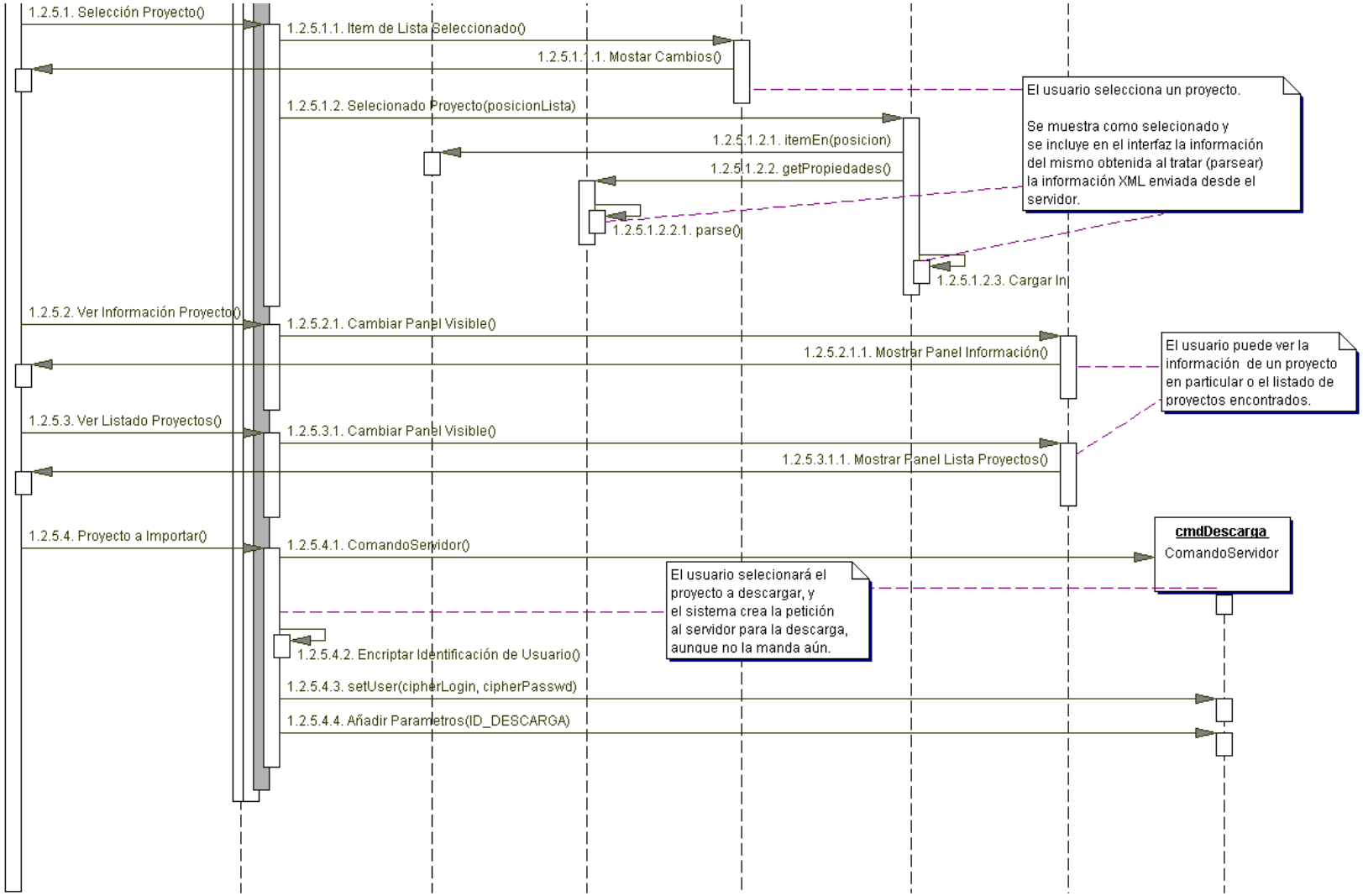

Figura 6.10. Secuencia Seleccionar Proyecto a Importar 2/2

6.4.1.7. Secuencia Importar Mecano (Cliente)

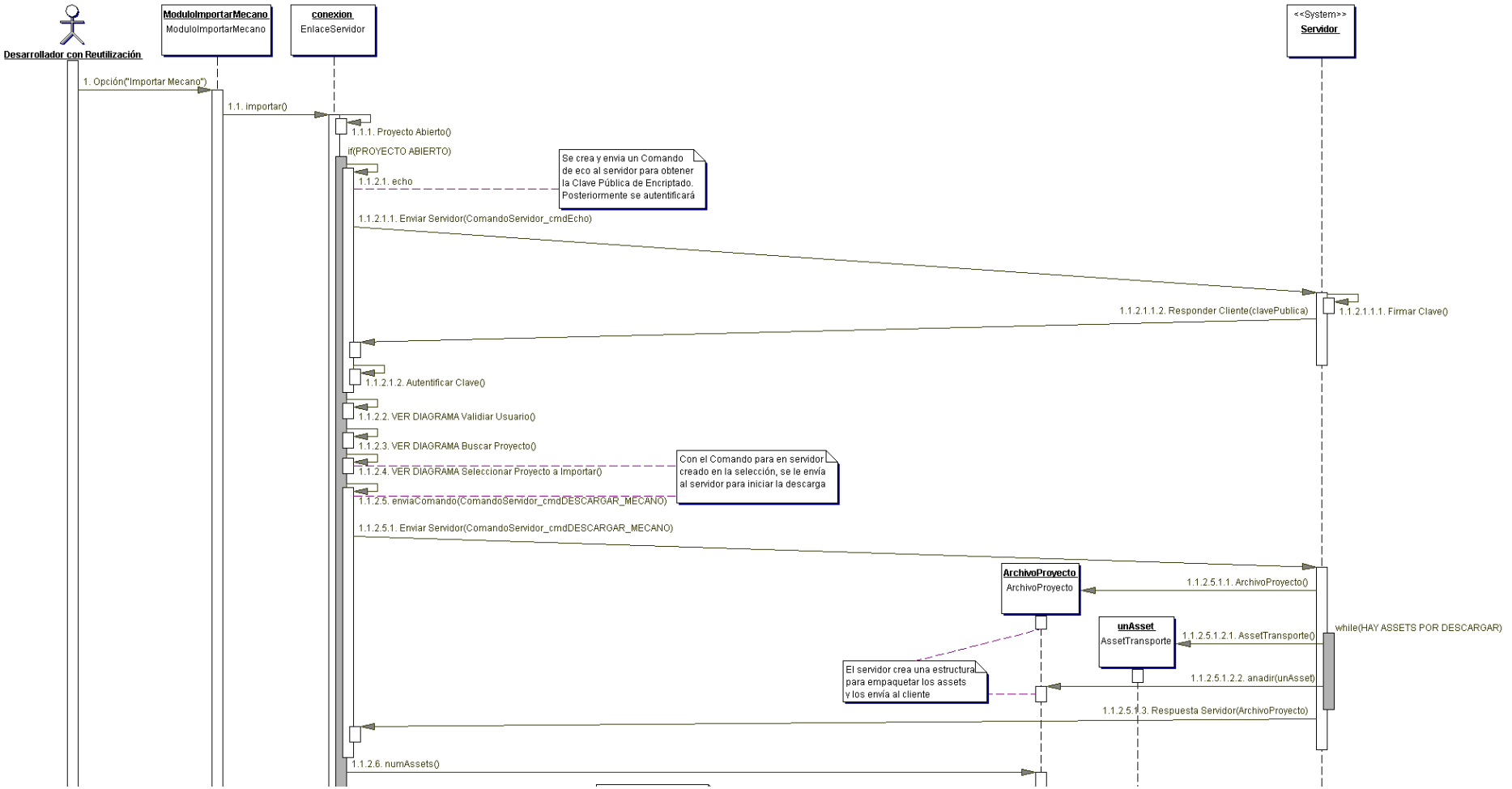

Figura 6.11. Secuencia Importar Mecano (Cliente) 1/2

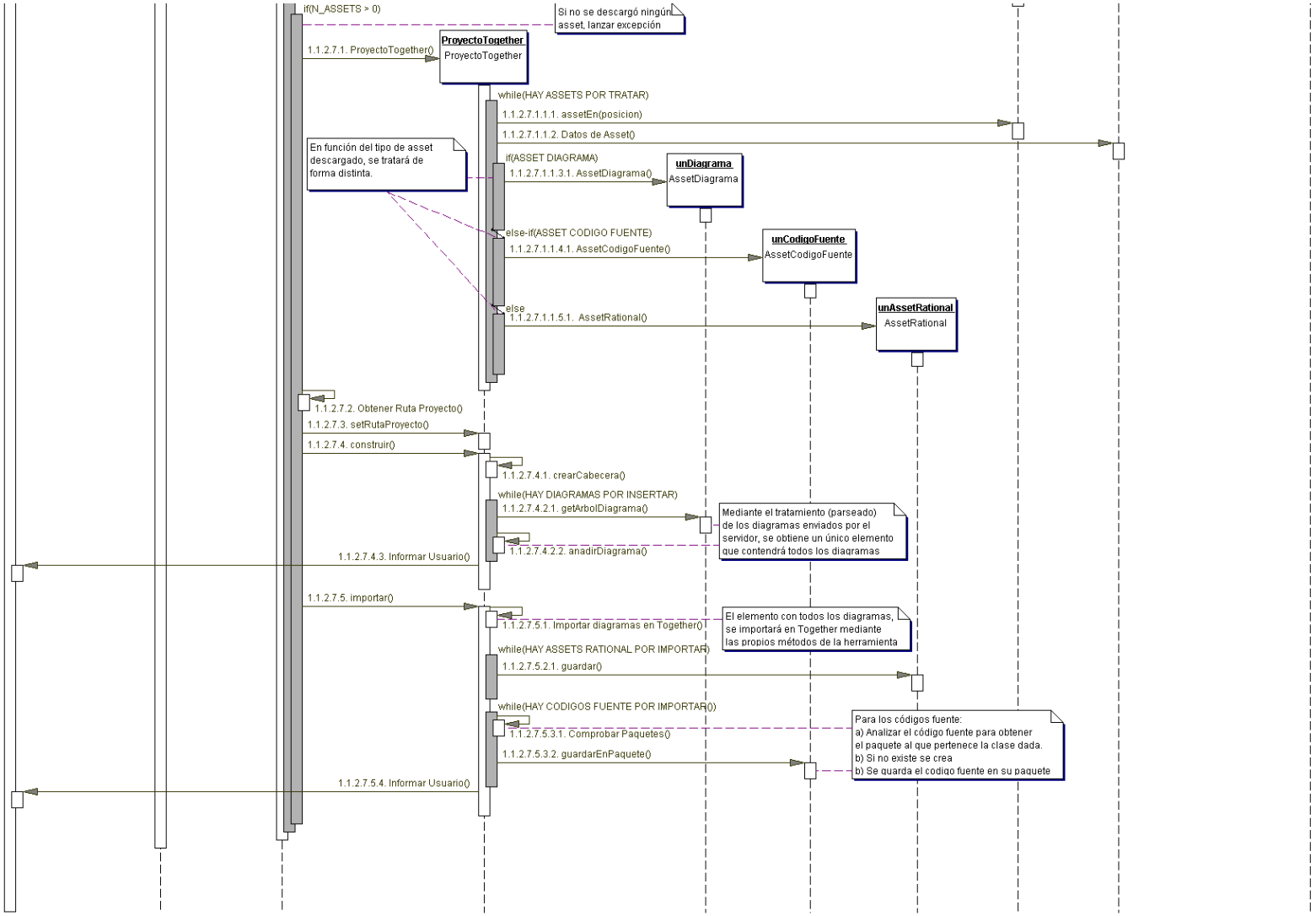

Figura 6.11. Secuencia Importar Mecano (Cliente) 2/2

6.4.1.8. Secuencia Importar Mecano (Servidor).

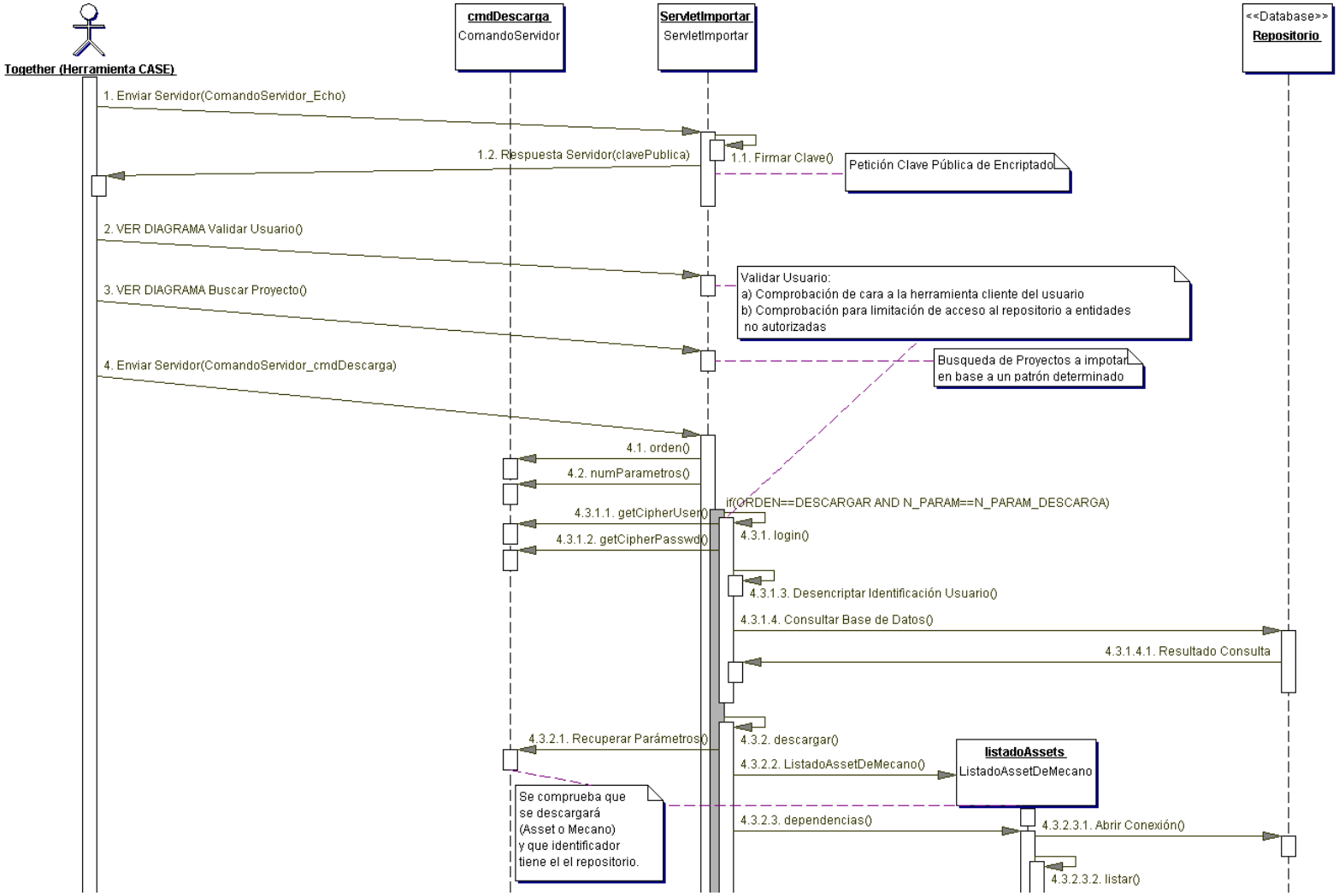

Figura 6.12. Secuencia Importar Mecano (Servidor) 1/2

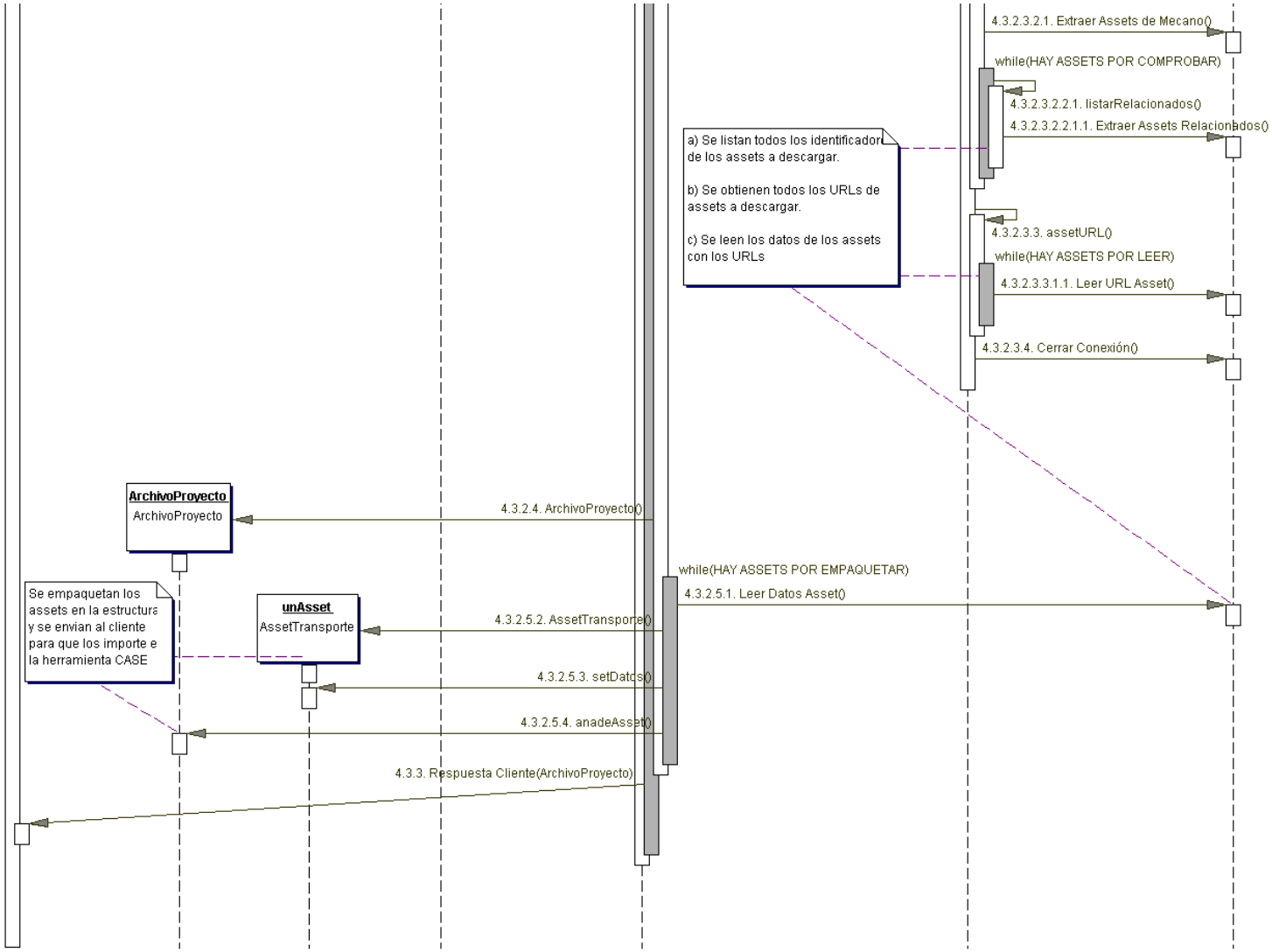

Figura 6.12. Secuencia Importar Mecano (Servidor) 2/2

6.4.1.9. Importar Asset (Cliente)

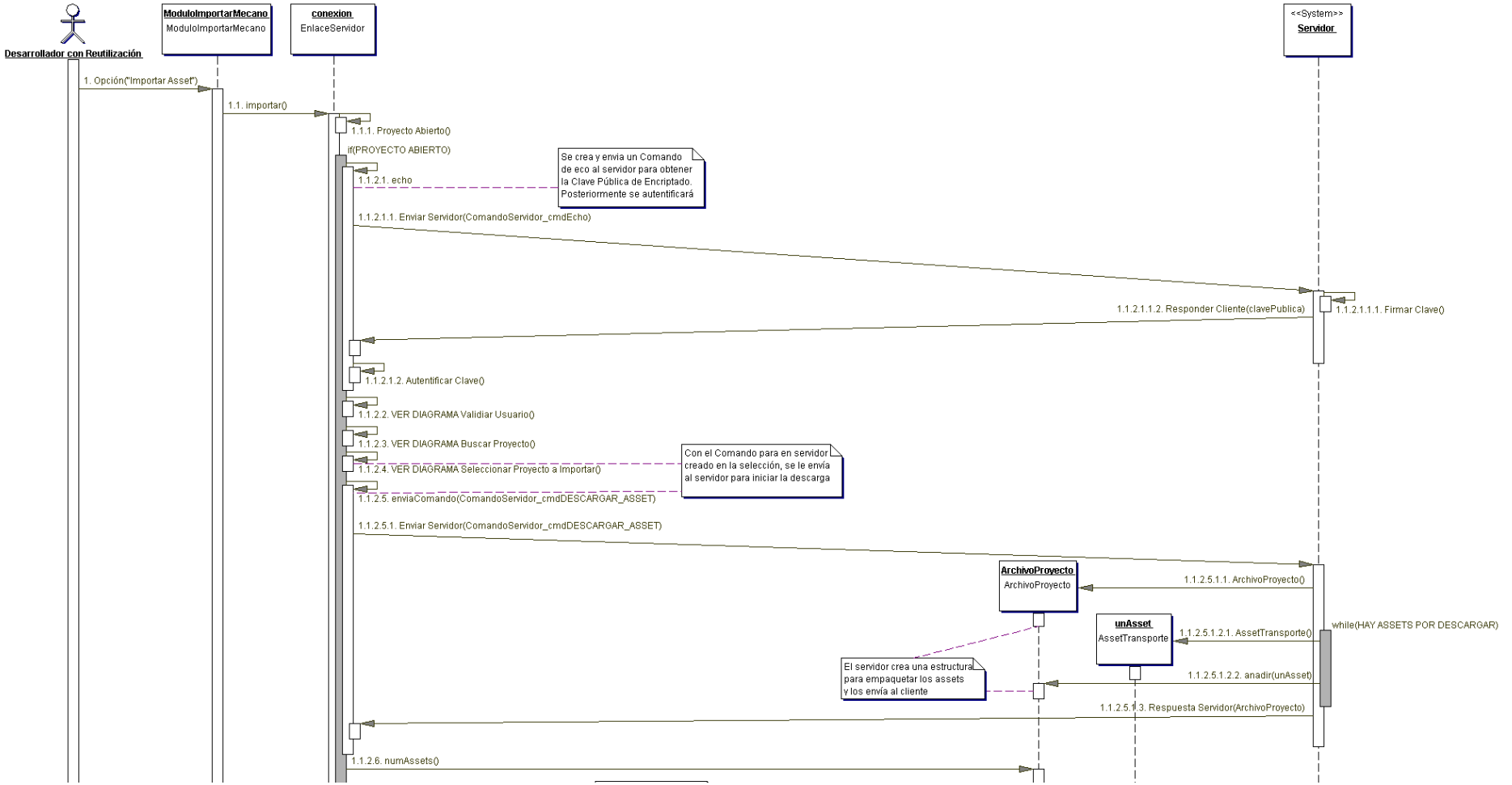

Figura 6.13. Secuencia Importar Asset (Cliente) 1/2

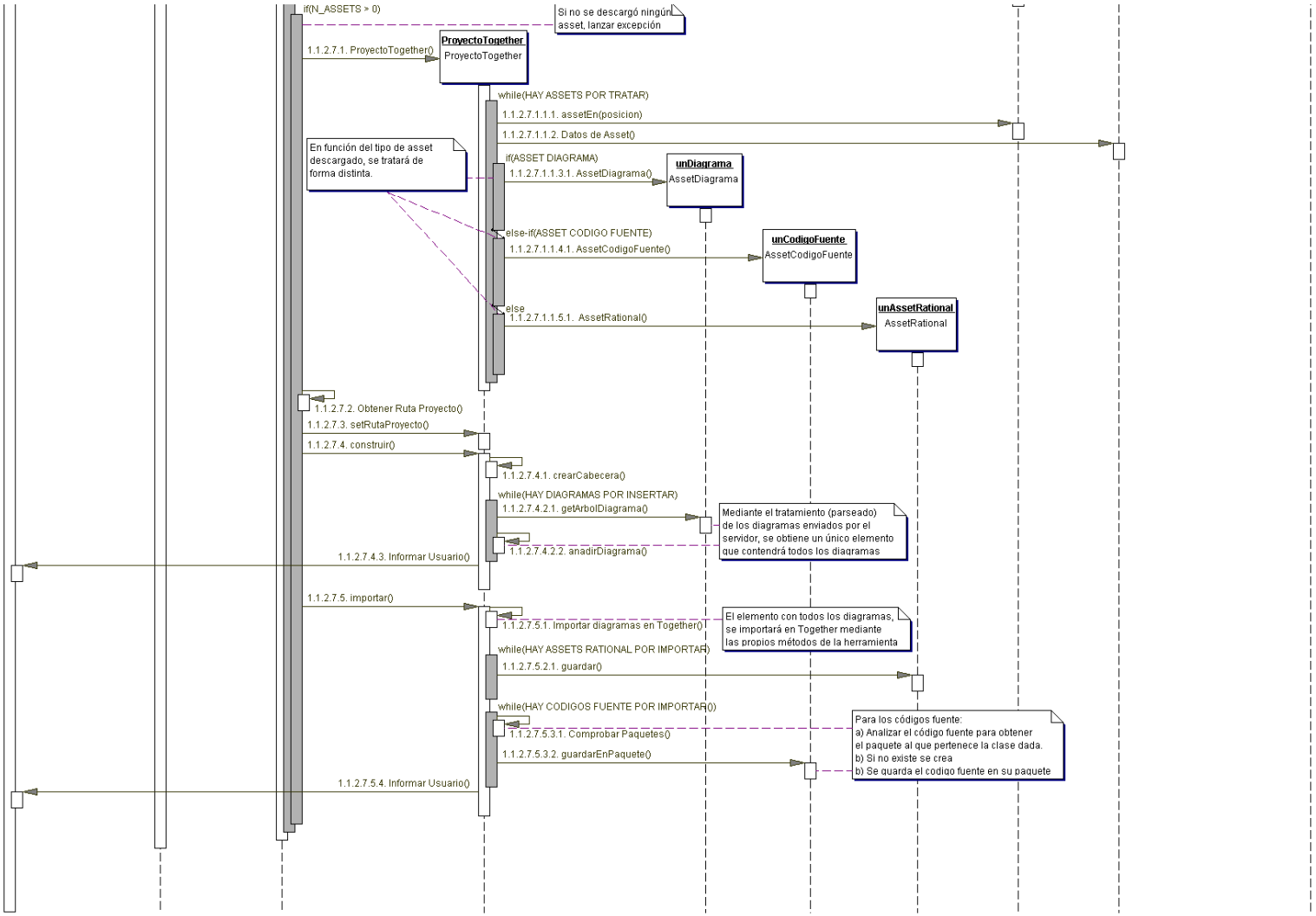

Figura 6.13. Secuencia Importar Asset (Cliente) 2/2

6.4.1.10. Secuencia Importar Asset (Servidor).

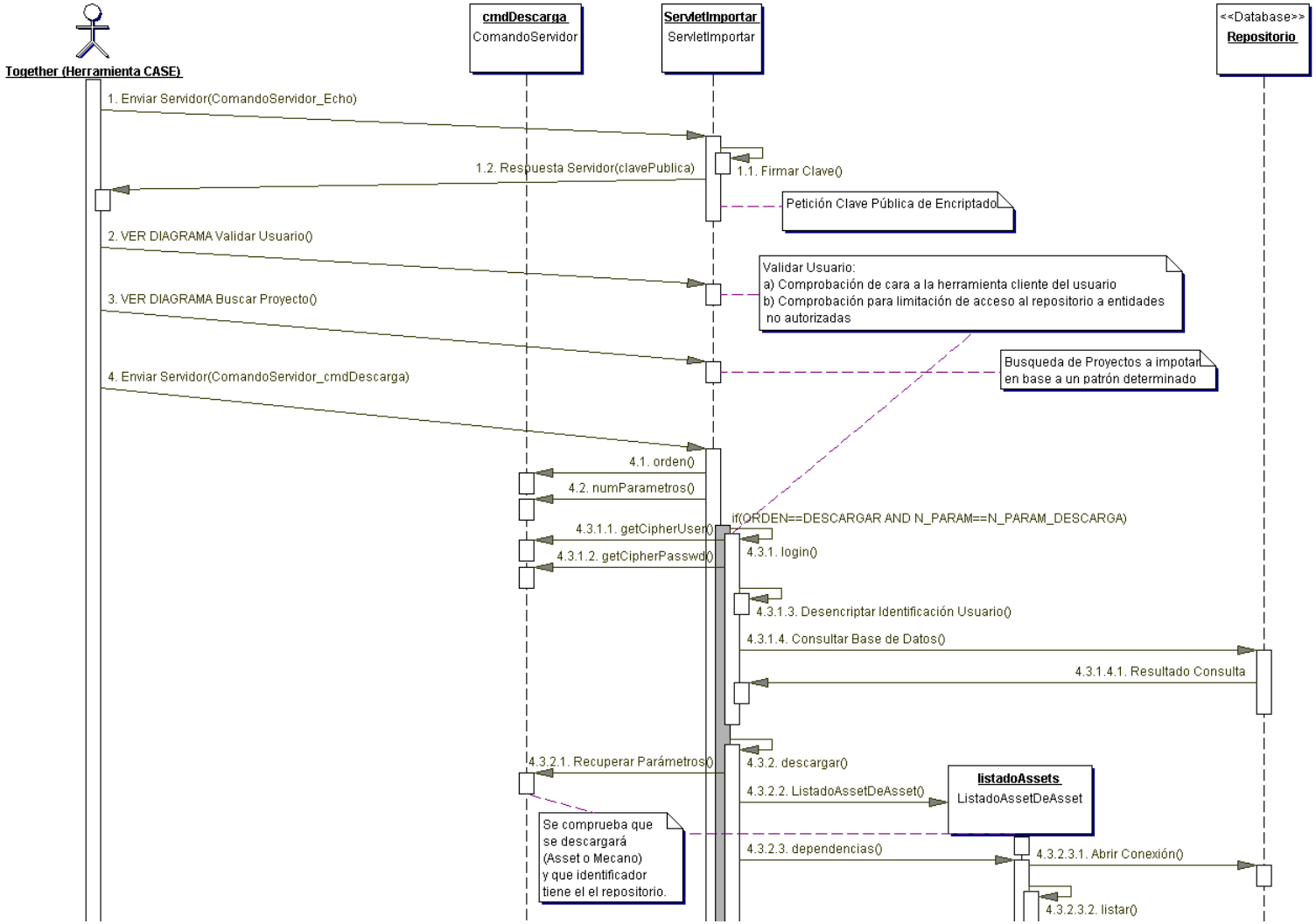

Figura 6.14. Secuencia Importar Asset (Servidor) 1/2

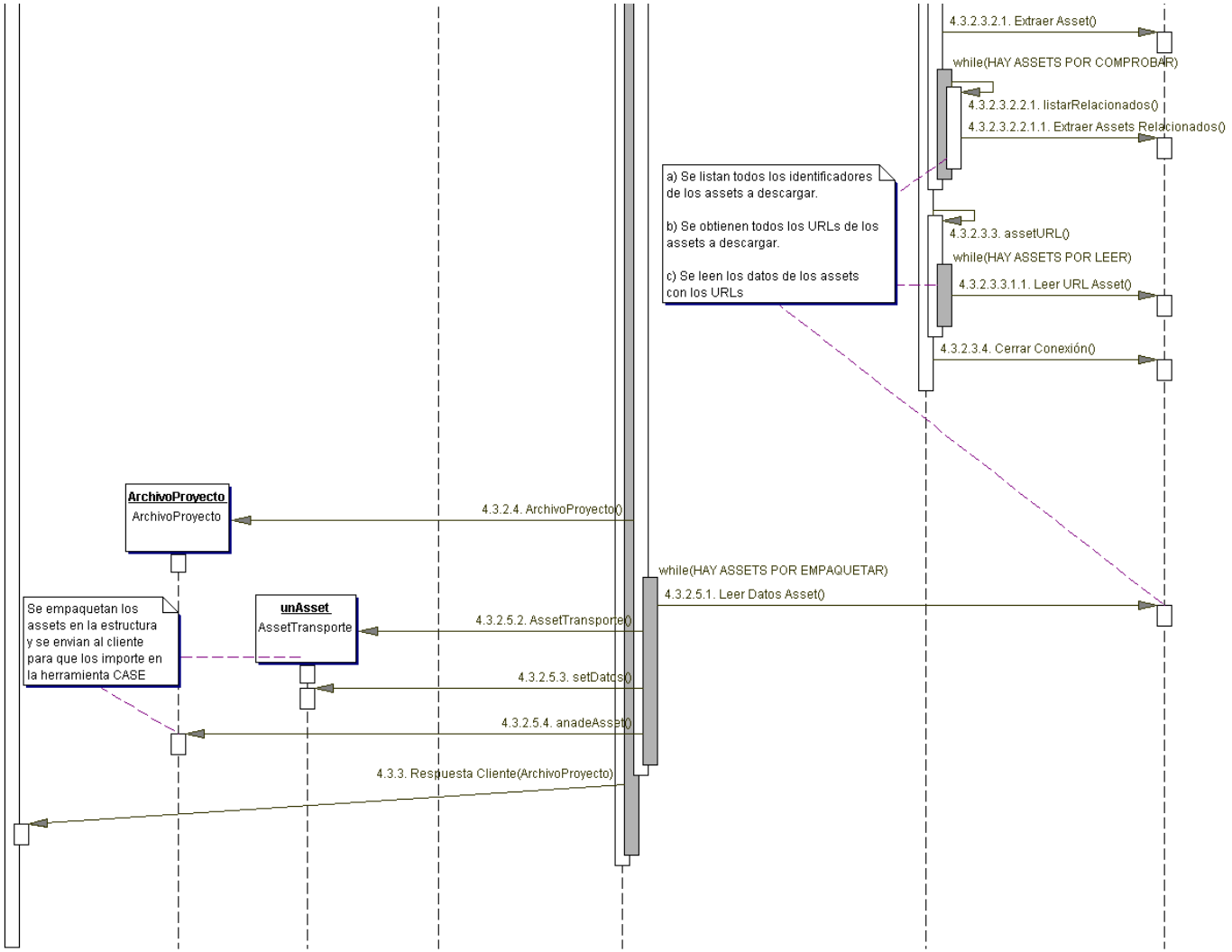

Figura 6.14. Secuencia Importar Asset (Servidor) 2/2

6.5. Diagramas de estados

6.5.1. Cliente

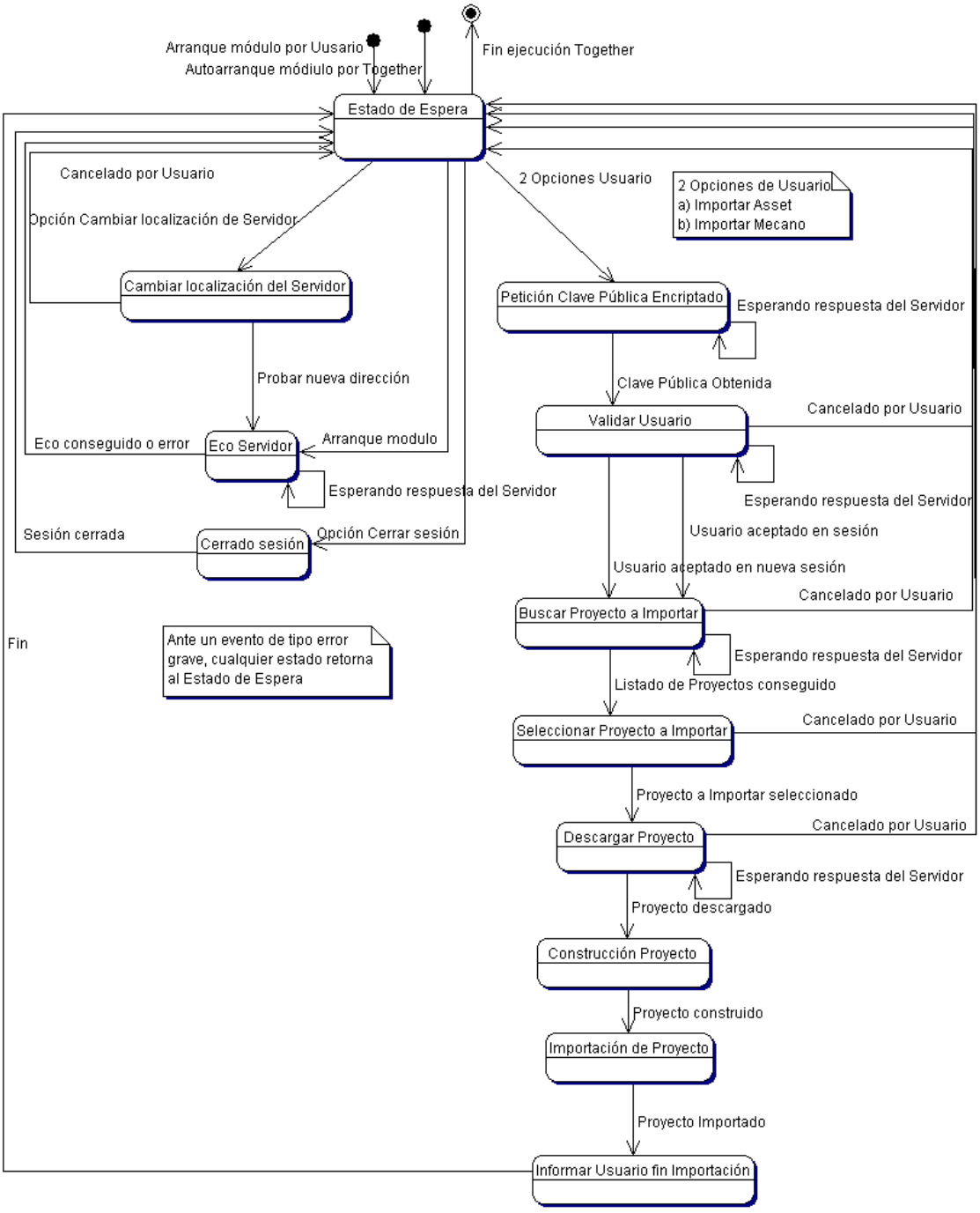

Figura 6.15.- Diagrama de Estados del Cliente

6.5.2. Servidor

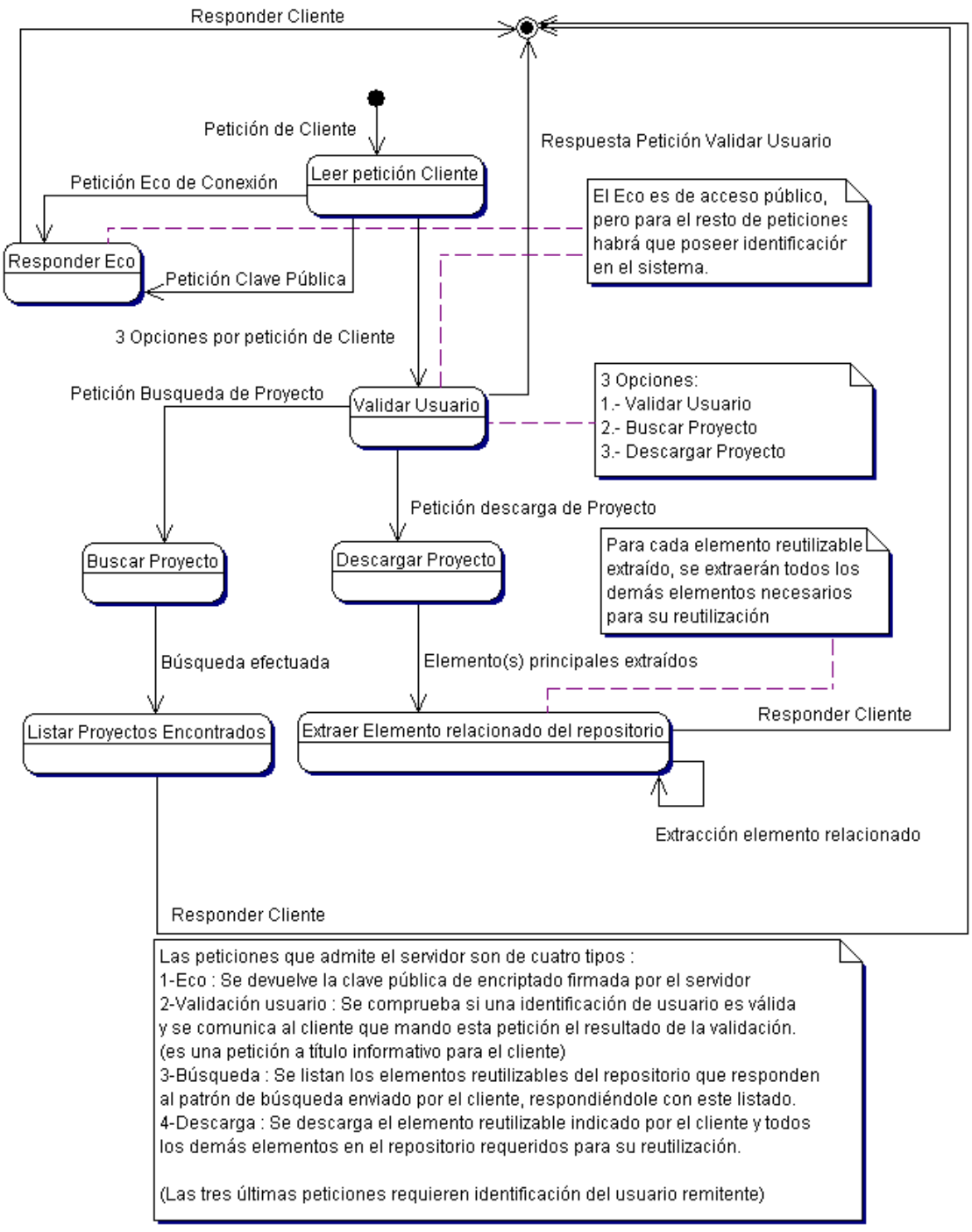

Figura 6.16.- Diagrama de Estados del Cliente

6.6. Diccionario de Clases

El diccionario de clases contiene la información referente a todas las clases que se van a implementar en el sistema, obtenido tras la fase de análisis de la aplicación.

A continuación se describen los diccionarios de clases de cada subsistema indicado en el apartado de arquitectura del sistema.

Se mostrará el diccionario de clases ordenándolas por su aparición en el Diagrama de Clases para facilitar su búsqueda durante la lectura del mismo.

6.6.1. Subsistema Cliente

6.6.1.1. ModuloImpotarMecano

Esta Clase constituye la raíz del módulo de Together que se ha de construir. Se integrará dentro de la interfaz de Together para mostrar al usuario las opciones que puede seleccionar para usar la funcionalidad del cliente y esa será toda su funcionalidad.

Atributo	Descripción	Tipo	Carácter
grupoImpotar	Esta variable creará un submenú de los menús de opciones de Together en el que se incluirán las funcionalidades del cliente de cara al usuario. El submenú será "Importar Proyecto Together" en el menú "Tools" de la herramienta CASE.	IdeCommandGroup	Privado
importaMecano	Esta variable es uno de los elementos del submenú de opciones desde la que se accede a la funcionalidad de "Importar Mecano".	IdeCommandItem	Privado
importaAsset	Esta variable es otro de los elementos del submenú de opciones desde la que se accede a la funcionalidad de "Importar Asset".	IdeCommandItem	Privado
logOut	Esta variable es otro de los elementos del submenú de opciones desde la que se accede a la funcionalidad de "Cerrar sesión".	IdeCommandItem	Privado
servidorURL	Esta variable es otro de los elementos del submenú de opciones desde la que se accede a la funcionalidad de "Cambiar localización del Servidor".	IdeCommandItem	Privado
acercaDe	Esta variable es otro de los elementos del submenú de opciones en la que se mostrará un mensaje informativo del sistema desarrollado. Constituirá el elemento de menú	IdeCommandItem	Privado

	"Acerca de..."		
paginaImportar	Será una página en la ventana de mensajes de *Together* en la que se mostrarán los mensajes generados por el módulo creado para Together.	IdeMessagePage	Privado Estático
conexion	Este atributo realizará toda la funcionalidad del cliente, petición de datos al usuario, búsquedas, etc, y la más importante, importar elementos reutilizables en Together desde el repositorio.	EnlaceServidor	Privado

Método	Descripción	Carácter
run()	Es el método llamado al arrancar manualmente un usuario el funcionamiento del módulo desarrollado para Together. Su cometido será la creación de los submenús de opciones desde los que se podrá acceder a la funcionalidad del cliente, y la comprobación de que la conexión con el servidor es posible. • Parámetros: El contexto actual • Devuelve: Ninguno	Público
autorun()	Es el método llamado al arrancar funcionamiento del módulo desarrollado pata Together en el arranque de la propia herramienta. Su cometido será la creación de los submenús de opciones desde los que se podrá acceder a la funcionalidad del cliente, y la comprobación de que la conexión con el servidor es posible. • Parámetros: El contexto actual • Devuelve: Ninguno	Público
construirMenus()	Esté método crea todos los menús de opciones necesarios para acceder ala funcionalidad del cliente • Parámetros: Ninguno • Devuelve: Ninguno	Privado
ventanaAcercaDe()	Muestra una ventana con información relativa al sistema (nombre, autores, etc) de tipo "Acerca de..." • Parámetros: Ninguno • Devuelve: Ninguno	Privado
ventanaLogOut()	Permite acceder a la funcionalidad de cierre de sesión. • Parámetros: El contexto actual • Devuelve: Ninguno	Privado
cambiarServidor()	Esté método muestra una ventana al usuario para permitirle cambiar la dirección del servidor a la que el cliente se conectará. Con la nueva dirección o con la antigua si se cancelo la operación realizará una prueba de conexión con el servidor. • Parámetros: Ninguno • Devuelve: Ninguno	Privado

6.6.1.2. EnlaceServidor

Esta Clase llevará a cabo toda la funcionalidad del cliente, es decir, conexión con el servidor, validación de usuarios, descarga de elementos reutilizables e inserción en los proyectos de Together. Será el núcleo del subsistema cliente.

Atributo	Descripción	Tipo	Carácter
URLServidorGIRO	Esta variable contiene la dirección del servidor de este sistema, al que el cliente se conectará. Se inicializa por defecto a la dirección "http://marte.dcs.fi.uva.es"	String	Público Estático
serverPubKey	Esta variable es la clave pública de encriptado enviada desde el servidor.	PublicKey	Privado Estático
paginaImportar	Será una página en la ventana de mensajes de *Together* en la que se mostrarán los mensajes generados por el módulo creado para Together.	IdeMessagePage	Privado Estático
windowManager	Este método hace referencia al gestor de ventanas de la herramienta CASE Together que permitirá mostrar ventanas en la interfaz de Together al usuario.	IdeWindowManager	Privado Estático
NOUSER	Esta será la identificación de usuario NULA, es decir, la presente en el subsistema cliente cuando nadie se ha identificado aún o cuando se ha cerrado una sesión activa.	String	Privado Estático Final
NOPASSWD	Está será la contraseña de usuario NULA, es decir, la presente en el subsistema cliente cuando nadie se ha identificado aún o cuando se ha cerrado una sesión activa.	String	Privado Estático Final
validado	Este atributo indica si hay una sesión abierta en el cliente o no.	Boolean	Privado
userSI	Este atributo mantiene la identificación de un usuario del sistema que se identificó correctamente y aún no ha cerrado su sesión.	String	Privado
passwdSI	Este atributo mantiene la contraseña de un usuario del sistema que se identificó correctamente y aún no ha cerrado su sesión.	String	Privado
cph	Este atributo es el objeto que se utilizará para garantizar la privacidad de la identificación de usuario en las conexiones con el servidor mediante la encriptación de las comunicaciones.	Cipher	Privado Estático

miProycetoTogether	Este atributo será el elemento reutilizable que una vez descargado del servidor se integrará en un proyecto de Together.	ProyectoTogether	Privado

Método	Descripción	Carácter
EnlaceServidor()	Es el método llamado para crear una instancia de esta clase • Parámetros: Ninguno • Devuelve: Ninguno	Público
logOut()	Es el método llamado para cerrar una sesión de usuario en el cliente. Si no hubiera sesión abierta, no haría nada. • Parámetros: Ninguno • Devuelve: Ninguno	Público
echo()	Este método realizará una prueba de comunicación con el servidor, tras la cual se obtendrá la clave pública de encriptado del servidor (actualizada y autentificada). Cualquier error en la comunicación o autentificación invalidará la clave para garantizar así la seguridad. • Parámetros: Ninguno • Devuelve: Ninguno	Público
importar()	Con este método en con el que se logrará realizar la funcionalidad del cliente. Al llamarlo, realizará la serie de acciones definidas en los apartados anteriores de diseño y análisis para validar lograr importar un elemento reutilizable en un proyecto Together. • Parámetros: (String) Tipo de elemento reutilizable a importar. • Devuelve: Ninguno	Público
enviaComando()	Permite enviar una orden o petición al servidor y devuelve la contestación del servidor. NOTA: Las peticiones de ECO (petición Clave Pública) tendrán una estructura especial, por lo que serán tratadas con el método echo() no con éste. • Parámetros: (ComandoServidor) Orden o petición a enviar al servidor. • Devuelve: (Object) Objeto devuelto como contestación por el servidor	Privado
login()	Este método pedirá mediante una ventana gráfica en la interfaz de Together, los datos de identificación al usuario (si este no se identificó antes), dejando en cualquier caso una petición en el sistema para el servidor (nula o no). • Parámetros: Ninguno • Devuelve: (ComandoServidor) Petición de tipo "LOGIN" o "Validación de usuario" con la identificación encriptada de usuario para el servidor.	Privado
buscar()	Este método solicitará mediante una ventana gráfica al usuario que indique el patrón de búsqueda que ajuste a los elementos reutilizables que desea importar en un proyecto Together. Con el patrón obtenido, creará una petición para servidor (que dejará en el subsistema cliente) con dicho patrón	Privado

	• Parámetros: (String) Tipo de elemento reutilizable a buscar (Asset o Mecano) • Devuelve: (ComandoServidor) Petición de tipo "BUSQUEDA" con la identificación encriptada de usuario (presente anteriormente a su llamada en el sistema) y el patrón de búsqueda para el servidor.	
seleccionar()	Este método muestra al usuario un listado (detallado o no deseo del usuario) con los elementos reutilizables del tipo y características (que se ajustan a un patrón dado) obtenido del servidor a petición del cliente. Tras la selección definitiva del usuario de uno de ellos, creará una petición de tipo "DESCARGA" para el servidor, en la que incluirá además de la identificación encriptada del usuario, el tipo de elemento a descargar y su identificador UNIVOCO (obtenido del propio servidor) en el repositorio. • Parámetros: (ResultadoBusqueda) Listado de elementos reutilizables hallados por el servidor a petición del cliente. • Parámetros: (String) Tipo de elemento reutilizable a seleccionar (Asset o Mecano). • Devuelve: (ComandoServidor) Petición de tipo "LOGIN" o "Validación de usuario" con la identificación de usuario para el servidor.	Privado

6.6.1.3. ModuloMecanoException

Este Clase es una excepción definida en el cliente para el tratamiento de los errores generados por las condiciones particulares de funcionamiento del cliente.

Método	Descripción	Carácter
ModuloMecanoException()	Es el método llamado para crear una instancia de esta clase • Parámetros: (String) Mensaje que define el motivo de lanzamiento de la excepción • Devuelve: Ninguno	Público

6.6.1.4. ConstantesParser

Esta Clase contiene las definiciones de una serie de Constantes (para este sistema en particular) de uso reiterado para el parseado de documentos XMI.

Atributo	Descripción	Tipo	Carácter
DIAGRAMA	Contiene el nombre del TAG que en XMI (formato Unisys) define la representación de los diagramas.	String	Público Estático Final
CONTENIDO	Contiene el nombre del TAG que en XMI define el nodo de la estructura XML que contiene todas las definiciones de los elementos del documento XMI.	String	Público Estático Final

PAQUETE	Contiene el nombre del TAG que en XMI define los paquetes de un documento XMI.	String	Público Estático Final
ELEMENTO	Contiene el nombre del TAG que en XMI representa de elementos del modelo.	String	Público Estático Final
TIPO_DATO	Contiene el nombre del TAG que en XMI define los tipos de datos.	String	Público Estático Final
EXTENSIONES	Contiene el nombre del TAG que en XMI define el nodo del documento que contiene las extensiones (en formato Unisys, las extensiones contienen las presentaciones de diagramas).	String	Público Estático Final
TOGETHER_CONTENIDO	Contiene el identificador que Together proporciona al TAG CONTENIDO (definido antes) raiz al exportar un proyecto a XMI.	String	Público Estático Final
TOGETHER_EXTENSION	Contiene el identificador que Together proporciona al TAG EXTENSION (definido antes) raiz al exportar un proyecto a XMI.	String	Público Estático Final

6.6.1.5. ProyectoTogether

Esta Clase representa un elemento reutilizable con todos los elementos reutilizables necesarios para su reutilización y una serie de métodos que le permiten importarse al proyecto de Together activo en ese momento.

Atributo	Descripción	Tipo	Carácter
listaDeAssets	Este atributo referenciará a un array de los assets de los que se compondrá el proyecto Together a importar.	AssetGenerico[]	Privado
proyXMI	Este atributo es el árbol documento XML donde se reunirán todos los assets diagramas (XMI) para su importación.	Document	Privado
proyectoXML	Es el fichero en el que se volcará el contenido del árbol antes nombrado, para poderlo importar.	File	Privado
numAssets	Es el número de assets de los que se compone le proyecto a importar	Int	Privado
diaClasses	Este atributo es un listado de los nombres (completos) todas las clases que son referenciadas (aparecen) en los diagramas de un proyecto Together. Facilitará la	Vector:String	Privado

	importación de códigos fuente.		
paginaImportar	Este atributo representa una página en la ventana de mensajes de Together.	IdeMessagePage	Privado Estático
miParser	Este atributo es un analizador sintáctico que se usará para obtener el paquete al que pertenece una clase (asset código fuente) que no es referenciada por los diagramas.	PackageJavaParser	Privado Estático

Método	Descripción	Carácter
ProyectoTogether()	Es el método llamado para crear una instancia de esta clase • Parámetros: (ArchivoProyecto) Una estructura con todos los assets del proyecto a importar en Together. • Devuelve: Ninguno	Público
construir()	Es el método llamado para unificar todos los diagramas XMI de los que se componga el proyecto a importar, en un único documento XML. • Parámetros: Ninguno • Devuelve: Ninguno	Público
importar()	Este método realizará la importación de todos los assets del proyecto a Together. • Parámetros: Ninguno • Devuelve: Ninguno	Público
crearCabecera()	Con este método se crea la cabecera común a todos los ficheros XMI (Unisys) que Together utiliza para el intercambio de datos (importación-exportación) • Parámetros: Ninguno • Devuelve: Ninguno	Privado
annadirDiagrama()	Añade un diagrama XMI al documento XMI que englobará a todos los diagramas del proyecto a importar. NOTA: Requiere de la existencia previa de una estructura básica (cabecera) sobre la que añadir un diagrama • Parámetros: (Document) Diagrama XMI a insertar en el todo. • Devuelve: Ninguno	Privado
buscar()	Este método permite buscar un elemento con un identificador dado en un documento XMI. • Parámetros: (Node) Nodo del documento desde el que empezar a buscar. • Parámetros: (String) Identificador (clave) a buscar. • Devuelve: (Element) Elemento que posee dicho identificador o NULL si no lo encontró.	Privado
procesaPaquete()	Este método permite recorrer las estructuras de un proyecto de Together en busca de los nombres completos (cualificados) de las clases que forman parte de ellos. • Parámetros: (SciPackage) Paquete de código fuente a partir del que buscar.	Privado

	• Devuelve: Ninguno.	

Propiedad	Descripción	Tipo	Carácter
rutaProyecto	Esta propiedad contiene la ruta completa del proyecto Together sobre el que se importarán los elementos reutilizables obtenidos del servidor.	String	Lectura Escritura

6.6.1.6. AssetGenerico

Esta Clase Abstracta se utiliza para acceder de manera uniforme a unas propiedades comunes a todos los tipos de Assets que se importarán en un proyecto Together.

Propiedad	Descripción	Tipo	Carácter
tipo	Esta propiedad contiene el tipo de asset del que se trata.	Int	Lectura Escritura
nombre	Esta propiedad contiene el nombre del asset.	String	Lectura Escritura

6.6.1.7. AssetDiagrama

Esta Clase representa a los assets de tipo diagrama (archivos XML representado documentos XMI), y contiene los métodos necesarios para su tratamiento.

Atributo	Descripción	Tipo	Carácter
datos	Este atributo contiene un documento XML sin parsear (con etiquetas)	String	Privado

Método	Descripción	Carácter
AssetDiagrama()	Es el método llamado para crear una instancia de esta clase • Parámetros: (String) El documento XML que representa el asset en bruto (sin parsear). • Devuelve: Ninguno	Público

Propiedad	Descripción	Tipo	Carácter
arbolDiagrama	Esta propiedad contiene el documento XMI parseado en estructura de árbol para su fácil manejo.	Document	Lectura

6.6.1.8. AssetCodigoFuente

Esta Clase representa a un asset de tipo código fuente, y los métodos necesarios para su utilización

Método	Descripción	
AssetCodigoFuente()	Es el método llamado para crear una instancia de esta clase • Parámetros: (String) El código fuente en bruto. • Devuelve: Ninguno	Público
guardarEnPaquete()	Este método permite guardar el código fuente en un archivo situado en un paquete determinado. • Parámetros: (String) Ruta del paquete y archivo donde se debe guardar el código fuente. • Devuelve: Ninguno	Público

Propiedad	Descripción	Tipo	Carácter
datos	Esta propiedad contiene el código fuente en bruto para su inserción en editores del API de Together.	String	Lectura

6.6.1.9. AssetRational

Esta Clase representa a los assets Rational ROSE (formato MDL) que Together permite importar de forma individual.

Atributo	Descripción	Tipo	Carácter
datos	Este atributo contiene el asset Rational en bruto.	String	Privado

Método	Descripción	Carácter
AssetRational()	Es el método llamado para crear una instancia de esta clase • Parámetros: (String) El contenido del asset Rational en bruto. • Devuelve: Ninguno	Público
guardar()	Este método permite guardar el asset Rational en la ruta deseada para su posterior importación a Together (opcional y voluntaria) • Parámetros: (String) Ruta deseada en la que se colocará el asset (sólo directorio). • Devuelve: Ninguno	Público

6.6.1.10. Paquete RenderSwing

6.6.1.10.1. ItemListCellRenderer

Esta Clase se encargará de la escucha de eventos en un elemento lista para interactuar con el usuario.

Atributo	Descripción	Tipo	Carácter
flecha	Es el icono imagen que se situará en la interfaz gráfica junto a un elemento de un listado.	ImageIcon	Privado
flechaSEL	Es el icono imagen que se situará en la interfaz gráfica junto a un elemento de un listado que haya sido seleccionado por el usuario.	ImageIcon	Privado

Método	Descripción	Carácter
getListCellRendererComponent()	Es el método llamado para dibujar los elementos de un listado en la interfaz. • Parámetros: (JList) La lista a mostrar. • Parámetros: (Object) El valor a mostrar en una posición de la lista. • Parámetros: (int) La posición de la lista a mostrar. • Parámetros. (boolean) True si el elemento está seleccionado en la lista. • Parámetros: (boolean) True si el elemento seleccionado tiene el foco de la interfaz.. • Devuelve: (Component) El item del listado para la interfaz.	Público

6.6.1.10.2. infoMecano

Esta Clase se encargará de la escucha de eventos en un elemento lista para cargar en la interfaz la información relevante de cada elemento seleccionado.

Atributo	Descripción	Tipo	Carácter
resultados	Este atributo en el listado de elementos de la lista del interfaz.	ResultadoBusqueda	Privado
TAM_INFO	Es la mayor longitud que será mostrada de la cadena de caracteres que describe una propiedad de un elemento del listado	Int	Privado Estático Final
informaciones	Es un array de las propiedades del objeto seleccionado del listado que se mostrará en la interfaz	JLabel[]	Privado

identi	Es una tabla Hash para conseguir la posición de un elemento del listado a partir de su valor	Hashtable	Privado
tipoEle	El tipo de elemento reutilizable que se está listando.	String	Privado

Método	Descripción	Carácter
infoMecano()	Es el método llamado para crear una instancia de esta clase. • Parámetros: (ResultadoBusqueda) El listado de elementos a mostrar. • Parámetros: (Hashtable) La tabla de acceso a posiciones por contenido. • Parámetros: (JLabel[]) La parte de la interfaz a modificar. • Parámetros: (String) El tipo de elemento reutilizable a mostrar. • Devuelve: Ninguno	Público
valueChanged()	Es el método llamado al seleccionar un elemento del listado de la interfaz., para que actualice las propiedades mostradas del elemento reutilizable seleccionado. • Parámetros: (ListSelectionEvent) El evento que desencadenó su llamada. • Devuelve: Ninguno	Público

6.6.1.10.2. cambiaTab

Esta Clase se encargará de la escucha de eventos en una parte de la interfaz del cliente para intercambiar que parte de la interfaz se mostrará al usuario y cual no.

Atributo	Descripción	Tipo	Carácter
fwd	Este atributo indica en que sentido se intercambiarán en la interfaz los paneles de listado e información.	Boolean	Privado
itemTab	Es la parte del interfaz que se desea intercambiar.	JTabbedPane	Privado

Método	Descripción	Carácter
cambiaTab()	Es el método llamado para crear una instancia de esta clase. • Parámetros: (JTabbedPane) Los paneles de la interfaz e intercambiar. • Parámetros: (boolean) El sentido del cambio (hacia delante, hacia atrás). • Devuelve: Ninguno	Público
actionPerformed()	Es el método llamado por la interfaz para que esta clase ejecute su funcionalidad • Parámetros: (ActionEvent) El evento que desencadenó su llamada. • Devuelve: Ninguno	Público

6.6.2. Subsistema Servidor:

6.6.2.1. ServletImportar

Esta Clase representa el servidor que recibirá y procesará todas las peticiones que el cliente le envíe. Sobre ella recaen casi todas las responsabilidades del servidor.

Atributo	Descripción	Tipo	Carácter
XMI	Este atributo es la extensión de los archivos assets diagramas del repositorio.	String	Público Estático Final
MDL	Este atributo es la extensión de los archivos assets de rational ROSE del repositorio.	String	Público Estático Final
codigoFuente	Este atributo es la extensión de los archivos assets códigos fuente del repositorio.	String	Público Estático Final
consBD	Es un conjunto de constantes que permiten la conexión a la base de datos del repositorio.	SGBDConstantes	Público Estático Final
tiposAsset	Es una tabla Hash para asociar a cada identificador de una tabla en la base de datos una representación textual de la misma.	Hashtable	Privado Estático
nvlsSeg	Es una tabla Hash para asociar a cada identificador de una tabla en la base de datos una representación textual de la misma.	Hashtable	Privado Estático
idiomas	Es una tabla Hash para asociar a cada identificador de una tabla en la base de datos una representación textual de la misma.	Hashtable	Privado Estático
nvlsAbs	Es una tabla Hash para asociar a cada identificador de una tabla en la base de datos una representación textual de la misma.	Hashtable	Privado Estático
miPubKey	Es la Clave Pública de encriptado del servidor.	PublicKey	Privado Estático
miPrivKey	Es la Clave Privada de encriptado del servidor. Le servirá para leer las identificaciones de los usuarios clientes y para firmar la Clave	PrivateKey	Privado Estático

	Pública al distribuirla.		
cph	Es el objeto que se encargará del encriptado de las comunicaciones.	Cipher	Privado Estático

Método	Descripción	Carácter
init()	Es el método llamado para comenzar la ejecución de un servlet. • Parámetros: (ServletConfig) La configuración del servlet • Devuelve: Ninguno	Público
destroy()	Es el método llamado para descargar de memoria un servlet, con lo que termina su ciclo de vida • Parámetros: Ninguno • Devuelve: Ninguno	Público
doPost()	Es el método llamado cuando el servlet recibe una petición HTTP de tipo POST • Parámetros: (HttpServletRequest) La peticón que se hace al servlet. • Parámetros: (HttpServletResponse) La respuesta del servlet • Devuelve: Ninguno	Público
doGet()	Es el método llamado cuando el servlet recibe una petición HTTP de tipo GET. • Parámetros: (HttpServletRequest) La peticón que se hace al servlet. • Parámetros: (HttpServletResponse) La respuesta del servlet • Devuelve: Ninguno	Público
login()	Es el método empleado para comprobar la validez de la identificación enviada por el cliente que solicita una petición. • Parámetros: (ComandoServidor) La petición del cliente • Devuelve: Ninguno	Privado
buscar()	Este método se emplea para listar (a petición de un cliente) todos los elementos reutilizables del repositorio que respondan a un patrón de búsqueda. • Parámetros: (ComandoServidor) La petición del cliente • Devuelve: (ResultadoBusqueda) El listado de los elementos reutilizables que se ajustan al patrón de búsqueda.	Privado
descargar()	Este método es llamado para atender una petición de descarga de elementos reutilizables por parte de un cliente del servidor. • Parámetros: (ComandoServidor) La petición del cliente • Devuelve: (ArchivoProyecto) El elemento	Privado

	reutilizable a descargar y todos aquellos elementos adicionales del repositorio requeridos para su reutilización).	

6.6.2.2. ServletImportarException

Esta Clase representa una excepción que es lanzada por el servidor para tratar los errores particulares a su funcionamiento.

Método	Descripción	Carácter
ServletImportarException()	Es el método llamado para crear una instancia de esta clase • Parámetros: (String) Mensaje que define el motivo de lanzamiento de la excepción • Devuelve: Ninguno	Público

6.6.2.3. SGBDConstantes

Esta clase contiene una serie de constantes para acceder a la base de datos del repositorio. Proviene de un trabajo anterior realizado sobre reutilización en el que se definió esta clase para uniformizar el acceso de toadas las aplicaciones al repositorio.

Atributo	Descripción	Tipo	Carácter
CONFINGURATION_FILE_SGBD	Constante con la ruta del fichero de configuración para el acceso a al base de datos.	String	Privado Final
USERID_DEFAULT	Usuario por defecto de la base de datos.	String	Privado Estático Final
PASSWORD_DEFAULT	Contraseña del usuario por defecto de la base de datos.	String	Privado Estático Final
DRIVER_DEFAULT	El Driver JDBC por defecto de acceso a la base de datos	String	Privado Estático Final
DRIVERCLASS	El la clase del Driver JDBC por defecto de acceso a la base de datos	String	Privado Estático Final
HOST_DEFAULT	El HOST por defecto de acceso a la base de datos	String	Privado Estático Final
PORT_DEFAULT	El puerto por defecto de acceso a la base de datos	String	Privado Estático

			Final
SID_DEFAULT	El SID por defecto de acceso a la base de datos	String	Privado Estático Final
USERID	El usuario con el que se accederá a la base de datos	String	Privado Estático
PASSWORD	La contraseña del usuario con el que se accederá a la base de datos	String	Privado Estático
DRIVER	El Driver JDBC de acceso a la base de datos	String	Privado Estático
HOST	El HOST de acceso a la base de datos	String	Privado Estático
PORT	El puerto de acceso a la base de datos	String	Privado Estático
PROTOCOL	El protocolo de acceso a la base de datos	String	Privado Estático
SID	El SID de acceso a la base de datos	String	Privado Estático
CONNECTION	La conexión con la que se dará acceso a la base de datos	String	Privado Estático
config	Los elementos del fichero de acceso a la base de datos.	Properties	Privado
file	El stream de lectura del fichero de configuración	FileInputStream	Privado

Propiedad	Descripción	Tipo	Carácter
userId	Esta propiedad contiene el usuario con el que se accederá a la base de datos.	String	Lectura
password	Esta propiedad contiene el password del usuario con el que se accederá a la base de datos.	String	Lectura
driver	Esta propiedad contiene el driver JDBC con el que se accederá a la base de datos.	String	Lectura
host	Esta propiedad contiene el host con el que se accederá a la base de datos.	String	Lectura
protocol	Esta propiedad contiene el protocolo con el que se accederá a la base de datos.	String	Lectura
port	Esta propiedad contiene el puerto con el que se accederá a la base de datos.	String	Lectura

sid	Esta propiedad contiene el sid con el que se accederá a la base de datos.	String	Lectura
connection	Esta propiedad contiene la conexión con la que se accederá a la base de datos.	String	Lectura
driverClass	Esta propiedad contiene la calse del driver JDBC con el que se accederá a la base de datos.	String	Lectura

6.6.2.4. ListadoDeAssets

Esta Clase Abstracta proporciona una interfaz común para el listado de los assets relacionados de un asset en concreto o de un mecano. Define los atributos y métodos comunes a ambos casos.

Atributo	Descripción	Tipo	Carácter
sqlRel	Es la sentencia SQL de búsqueda en la base de datos.	String	Protegido
stmt	Es la sentencia SQL que realmente se ejecutará en la búsqueda.	PreparedStatement	Protegido
rs	Son los resultados de la búsqueda en la base de datos	ResultSet	Protegido
laRaiz	Este atributo representa el identificador raíz a partir del cual se listarán todos los elementos reutilizables del repositorio relacionados con él	String	Protegido
listadoURL	Es un listado de todos los URL de los elementos listados.	Vector:String	Protegido
listado	Es un listado con los identificadores de todos los elementos listados.	Vector:String	Protegido
conx	Es la conexión a través de la que se accede a la base de datos.	Connection	Protegido

Método	Descripción	Carácter
listar()	Este método listará los identificadores de todos los elementos del repositorio necesarios para la reutilización del elemento reutilizable principal que se desea descargar. • Parámetros: Ninguno • Devuelve: Ninguno	Protegido
cerrar()	Es el método llamado para cerrar la conexión con la base de datos. • Parámetros: Ninguno • Devuelve: Ninguno	Protegido
conectar()	Es el método llamado para abrir una conexión con la base de datos. • Parámetros: Ninguno	Protegido

	• Devuelve: Ninguno	
listarRelacionados()	Es el método llamado para obtener todos los elementos reutilizables del repositorio que con los que otro (parámetro del método) tiene una relación fuerte de dependencia. • Parámetros: (String) Identificador del elemento de la base de datos a comprobara sus dependencias • Devuelve: Ninguno	Protegido
assetURL()	Es el método empleado para listar los URLs de todos los elementos reutilizables a partir de los identificadores de los mismos presentes en un listado anterior. • Parámetros: Ninguno • Devuelve: Ninguno	Protegido
dependencias()	Este método se emplea para listar todos los URLs de los elementos reutilizables del repositorio necesarios para la reutilización de otro (incluido él mismo). • Parámetros: Ninguno • Devuelve: (Vector) Listado con los URLs solicitados.	Público

6.6.2.5. ListadoAssetDeMecano

Este Clase proporciona la funcionalidad de listar todos los assets relacionados (directa o indirectamente) de forma fuerte con los assets que componen un mecano.

Método	Descripción	Carácter
ListadoAssetDeMecano()	Es el método empleado para crear una instancia de esta Clase • Parámetros: (String) Identificador en la base de datos del Mecano a tratar. • Devuelve: Ninguno	Público
listar()	Este método listará los identificadores de todos los elementos del repositorio necesarios para la reutilización del Mecano que se desea descargar. • Parámetros: Ninguno • Devuelve: Ninguno	Protegido
dependencias()	Este método se emplea para listar todos los URLs de los assets del repositorio necesarios para la reutilización del Mecano. • Parámetros: Ninguno • Devuelve: (Vector) Listado con los URLs solicitados.	Público

6.6.2.6. ListadoAssetDeAsset

Esta Clase propopciona la funcionalidad de listar todos los assets relacionados (directa o indirectamente) de forma fuerte con un asset en concreto.

Método	Descripción	Carácter
ListadoAssetDeAsset()	Es el método empleado para crear una instancia de esta clase • Parámetros: (String) Identificador en el repositorio de asset a tratar. • Devuelve: Ninguno	Público
listar()	Este método listará los identificadores de todos los assets necesarios para la reutilización del asset principal que se desea descargar. • Parámetros: Ninguno • Devuelve: Ninguno	Protegido
dependencias()	Este método se emplea para listar todos los URLs de los assets del repositorio necesarios para la reutilización de otro (incluido él mismo). • Parámetros: Ninguno • Devuelve: (Vector) Listado con los URLs solicitados.	Público

6.6.3. Subsistema Comunicaciones

6.6.3.1. ConstantesComunicaciones

Esta Clase contiene una serie de constantes usadas tanto por cliente como por servidor de forma continuada y frecuente.

Atributo	Descripción	Tipo	Carácter
CMD_LOGIN	Constante que define una petición al servidor de validación de usuario.	String	Público Estático Final
CMD_DESCARGAR	Constante que define una petición al servidor de descarga de elementos reutilizables.	String	Público Estático Final
CMD_BUSCAR	Constante que define una petición al servidor de búsqueda de elementos reutilizables.	String	Público Estático Final
CMD_ECHO	Constante que define una petición al servidor de eco (obtención de Clave Pública de encriptado.	String	Público Estático Final
CMD_OK	Constante que define una respuesta afirmativa del servidor.	String	Público Estático

			Final
CMD_ERROR	Constante que define una respuesta negativa (error) del servidor.	String	Público Estático Final
MECANO	Constante que define elemento reutilizable de tipo mecano.	String	Público Estático Final
ASSET	Constante que define elemento reutilizable de tipo asset.	String	Público Estático Final
TIPO_DESCONOCIDO	Constante que define un asset de tipo desconocido.	Int	Público Estático Final
TIPO_RATIONAL	Constante que define un asset de tipo Rational ROSE.	Int	Público Estático Final
TIPO_DIAGRAMA	Constante que define un asset de tipo Diagrama XMI.	Int	Público Estático Final
TIPO_FUENTE	Constante que define un asset de tipo código fuente.	Int	Público Estático Final
SERVLET_SIGNATURE	Es una constante que define la firma del servidor en sus envíos de Claves Públicas de encriptado. Conocida por cliente y servidor permite verificar la autenticidad de dichas claves.	Byte[]	Público Estático Final

6.6.3.2. ComandoServidor

Esta Clase representa un orden o petición del cliente al servidor o una respuesta del tipo OK o ERROR del cliente al servidor. Proporciona una interfaz muy simple y clara para almacenar y recuperar parámetros, así como para incluir en las peticiones la identificación encriptada de su remitente.

Atributo	Descripción	Tipo	Carácter
orden	Es el tipo de petición u orden que define la petición en sí.	String	Privado
parametros	Son los parámetros de la petición.	Vector:String	Privado

Método	Descripción	Carácter
ComandoServidor()	Es el método empleado para crear una instancia de esta clase • Parámetros: (String) Tipo de orden que define la petición • Devuelve: Ninguno	Público
numParametros()	Este método permite conocer el número de parámetros de una petición. • Parámetros: Ninguno • Devuelve: (int) número de parámetros de la petición.	Público
anadeParametro()	Este método se emplea para añadir un parámetro a una petición. El orden de almacenamiento y recuperación es cronológico (ordenados por entrada) • Parámetros: (String) Parámetro a añadir • Devuelve: Ninguno.	Público
parametro()	Este método se emplea para recuperar un parámetro de una petición. El orden de almacenamiento y recuperación es cronológico (ordenados por entrada) • Parámetros: (int) Lugar del parámetro en la petición (0-indexado) • Devuelve: (String) Parámetro en dicha posición.	Público
comando()	Este método se emplea para recuperar el tipo de petición de una petición. • Parámetros: Ninguno • Devuelve: (String) Tipo de petición	Público
setUser()	Este método se emplea para añadir la identificación de usuario encriptada a una petición. • Parámetros: (byte[]) Identificación de usuario encriptada • Parámetros: (byte[]) Contraseña de usuario encriptada. • Devuelve: Ninguno.	Público

Propiedad	Descripción	Tipo	Carácter
cipherUser	Esta propiedad contiene la identificación encriptada del usuario que remite el comando.	Byte[]	Lectura
cipherPasswd	Esta propiedad contiene el password encriptado del usuario que remite el comando.	Byte[]	Lectura

6.6.3.3. ResultadoBusqueda

Esta Clase responde a la necesidad del empaquetado y acceso uniforme a una composición de objetos del clase ItemResultado.

Atributo	Descripción	Tipo	Carácter

resultados	Este atributo almacena los elementos ItemResultado de los que esta clase es una composición.	Vector:ItemResultado	Privado

Método	Descripción	Carácter
ResultadoBusqueda()	Es el método empleado para crear una instancia de esta clase. • Parámetros: Ninguno • Devuelve: Ninguno	Público
numResultados()	Este método permite conocer el número de elementos ItemResultado que contiene una instancia de esta clase. • Parámetros: Ninguno • Devuelve: (int) número de elementos ItemResultado	Público
annadir()	Este método se emplea para añadir un ItemResultado a la composición. El orden de almacenamiento y recuperación es cronológico (ordenados por entrada) • Parámetros: (ItemResultado) El elemento a almacenar. • Devuelve: Ninguno.	Público
itemEn()	Este método se emplea para recuperar un elemento ItemResultado de la composición. El orden de almacenamiento y recuperación es cronológico (ordenados por entrada) • Parámetros: (int) Lugar del elemento en la composición (0-indexado) • Devuelve: (ItemResultado) El elemento ItemResultado en dicha posición.	Público

6.6.3.4. ItemResultado

Esta Clase representa la información lógica de un elemento reutilizable, así como los métodos necesarios para un acceso cómodo a la vez que eficiente a esta información.

Atributo	Descripción	Tipo	Carácter
arbolProps	Es el documento XML parseado que contiene la información de un elemento reutilizable.	Document	Privado
props	Es el documento XML sin parsear (con etiquetas) que contiene la información de un elemento reutilizable.	String	Privado

Método	Descripción	Carácter
ItemResultado()	Es el método empleado para crear una instancia de esta clase	Público

	• Parámetros: (String) Identificador en el repositorio del elemento reutilizable. • Parámetros: (String) Nombre del elemento reutilizable en el erepositorio. • Devuelve: Ninguno	
cargar()	Este método carga en una instancia de esta clase la información lógica del elemento reutilizable que esta clase representa. • Parámetros: (String) Documento XML sin parsear con la información lógica del elemento reutilizable. • Devuelve: Ninguno	Público
parse()	Este método se emplea para parsear el documento XML con la información del elemento reutilizable y almacenar cada una de sus propiedades o características en la propiedad propiedades() de esta clase. • Parámetros: Ninguno • Devuelve: Ninguno.	Privado

Propiedad	Descripción	Tipo	Carácter
ID	Esta propiedad contiene la identificación unívoca del elemento reutilizable en el repositorio.	String	Lectura
nombre	Esta propiedad contiene el nombre del elemento reutilizable en el repositorio,	String	Lectura
propiedades	Esta propiedad es una tabla Hash que contiene todas loas propiedades o características del elemento reutilizable, a las que se puede acceder por el nombre de las mismas.	Hashtable	Lectura

6.6.3.5. ArchivoProyecto

Esta Clase responde a la necesidad del empaquetado y acceso uniforme a una composición de objetos del clase AssetTransporte. Se podría decir que es un proyecto Together en bruto, sin tratamiento alguno.

Atributo	Descripción	Tipo	Carácter
listaAssets	Este atributo almacena los elementos AssetTransporte de los que esta clase es una composición.	Vector	Privado

Método	Descripción	Carácter
ArchivoProyecto()	Es el método empleado para crear una instancia de esta clase	Público

	clase. • Parámetros: Ninguno • Devuelve: Ninguno	
numAssets()	Este método permite conocer el número de elementos AssetTransporte que contiene una instancia de esta clase. • Parámetros: Ninguno • Devuelve: (int) número de elementos AssetTransporte	Público
anadeAsset()	Este método se emplea para añadir un elemento AssetTransporte a la composición. El orden de almacenamiento y recuperación es cronológico (ordenados por entrada) • Parámetros: (AssetTransporte) El elemento a almacenar. • Devuelve: Ninguno.	Público
assetEn()	Este método se emplea para recuperar un elemento AssetTransporte de la composición. El orden de almacenamiento y recuperación es cronológico (ordenados por entrada) • Parámetros: (int) Lugar del elemento en la composición (0-indexado) • Devuelve: (AssetTransporte) El elemento AssetTransporte en dicha posición.	Público

6.6.3.6. AssetTransporte

Esta Clase contiene la representación física de un asset y dispone de los métodos necesarios para distinguir el tipo de asset así como su nombre y la obtención de los datos que dicha representación física encierra.

Método	Descripción	Carácter
AssetTransporte()	Es el método empleado para crear una instancia de esta clase • Parámetros: (String) Nombre del asset que almacenará. • Parámetros: (int) Tipo del asset que almacenará. • Devuelve: Ninguno	Público

Propiedad	Descripción	Tipo	Carácter
nombre	Nombre del asset que está almacenado.	String	Lectura
tipo	Tipo del asset que está almacenado.	Int	Lectura
datos	Representación física del asset almacenado	String	Lectura Escritura

7. DOCUMENTO DE IMPLEMENTACIÓN

7.1. Diagramas de componentes

Los diagramas de componentes muestran las dependencias entre componentes software, incluidos componentes de código fuente, de código binario o ejecutables. Por tanto, los componentes pueden existir en un tiempo de compilación, de enlazado o de ejecución e incluso en varios tiempos a la vez.

En este apartado se mostrarán los componentes d este sistema, así como su forma de ensamblado, describiendo especialmente además de los paquetes creados en el desarrollo del sistema, aquellos componentes de terceras personas usados para la elaboración del sistema, así como una breve descripción. Aquí el término "componente de terceras personas" incluyen tanto a la herramienta Together, como el motor gestor de servlets Jakarta-Tomcat (para Apache), parsers, seguridad, etc.

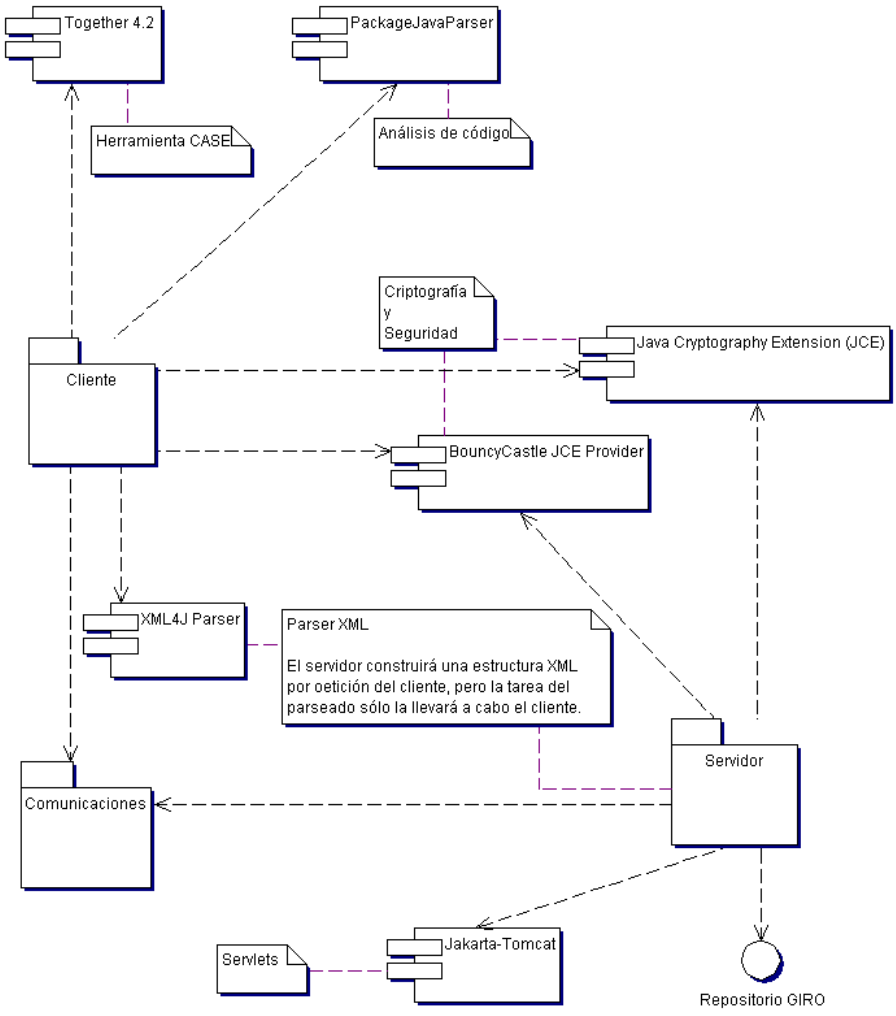

Figura 7.1. Diagrama de Componentes

7.1.1. La herramienta CASE *Together* y el subsistema Cliente.

La herramienta CASE Together es una aplicación multiplataforma escrita por completo en Java que permite el desarrollo de software orientado a objetos en múltiples lenguajes de programación, aunque esta orientada principalmente al desarrollo de software en Java.

La herramienta CASE supone algo más que un potente entorno de desarrollo integrado (IDE, *Integrated Depelovemnet Enviroment*) orientado a objetos, puesto que presenta de cara a los usuarios que la utilicen, la posibilidad de crear módulos propios para la herramienta.

Estos módulos quedan totalmente integrados en Together y permiten acceder a las funciones del API de Together, posibilitando desde simples personalizaciones de una función de Together, hasta la construcción de sistemas complejos aumenten en gran medida las posibilidades de uso de la herramienta (como es el caso, dotar a Together de desarrollo con reutilización).

A continuación se explicará la forma de desarrollar módulos Together y en particular las decisiones tomadas en el desarrollo del módulo de Together que responde al componente Cliente del sistema que se expone en este documento.

7.1.1.1 El API de *Together*.

El API de Together se puede descomponer en tres grandes interfaces de funcionalidad específica dentro de la herramienta CASE:

- Un interfaz de entorno de desarrollo integrado (*IDE*)
- Un interfaz de lectura escritura (*RWI*)
- Un interfaz de acceso al código fuente (*SCI*)

En una visión más detallada a cada uno de los interfaces se tiene:

IDE: A través de esta interfaz, se tiene acceso en modo de sólo lectura a la representación de alto nivel que de cara al usuario proporciona Together de un proyecto. En este interfaz también se encuentran todas aquellas funciones que permiten a los módulos generados por los usuarios de Together mostrar ventanas gráficas en el interfaz de usuario, mostrar mensajes, etc.

RWI: Permite acceder a más bajo nivel a la representación de un proyecto en Together, proporcionando acceso a la lectura y modificación de las estructuras de un proyecto representado en Together. Funciones típicas de este interfaz son las que dan acceso de forma individualizada (permitiendo modificaciones) a los diagramas del proyecto y a las entidades que los forman (notas, actores, casos de uso, etc.).

SCI: Este interfaz proporciona acceso al código fuente de un proyecto Together. A través de él se pueden añadir nuevos elementos al código fuente de forma fácil y uniforme. Funcionalidades típicas de este interfaz son el acceso en lectura escritura el código de un método, la inserción de nuevos atributos en una clase, el acceso a los

paquetes en que se divide el código fuente de un proyecto, e incluso a paquetes ya compilados (ZIP, JAR).

7.1.1.2. Descripción de los módulos de *Together*

La versatilidad de los módulos de Together es tal, que gran parte de la funcionalidad de Together está escrita en base a módulos de la propia herramienta, permitiendo incluso la inclusión de la funcionalidad de dichos módulos en las barras de herramientas de Together (Menú Tools).

En función del proceso de desarrollo que sufren los módulos de Together se pueden clasificar en la siguiente tipología:

- **Compilado**: Son módulos escritos en Java y que una vez compilados son ejecutados por la propia máquina virtual (JVM, *Java Virtual Machine*) sobre la que está siendo ejecutado Together.

- **Interpretado**: Son módulos escritos en TCL o Jphyton, que son ejecutados por intérpretes de la propia herramienta Together en tiempo de ejecución.

En cuanto a su manera de ser cargados en el sistema, los módulos de Together pueden dividirse en cuatro grandes grupos. Esta forma de ser cargados o llamados por la herramienta tendrá repercusión en su diseño e implementación como puede serse a continuación:

- **Módulo de Usuario**: El módulo se encuentra accesible a través del árbol de modulos de Together y ha de ser invocado mediante una orden explícita de usuario. Deben implementar la interfaz *IdeScript* que define un método *run()* para el comienzo de su ejecución.

- **Módulo de Usuario con Pre-Inicialización**: Es similar al módulo de usuario, pero debe definirse un método de inicio, porque dichos métodos son ejecutados antes de su llamada o invocación.

- **Módulo de Inicio**: Estos módulos son llamados en el arranque de la Together por la propia herramienta. Deben implementar la interfaz *IdeStartup* que define un método *autorun()* que será invocado por Together durante su proceso se arranque.

- **Módulo Activable**: Estos módulos combinan las prestaciones de los anteriores permitiendo tanto su carga manual en el sistema como su autoarranque con Together, pero además proporcionan la capacidad de ser desactivados y que liberen recursos al sistema. Deben implemetar la interfaz *IdeActivatable* que requiere la definición del método *shutdown()* para su desactivación.

Together impone una restricción tanto en el desarrollo, imponiendo la pertenencia de los módulos al paquete com.togethersoft.modules, o a alguno paquete

contenido dentro de este, así como en la localización física de los módulos, imponiendo que sean almacenados en el directorio:

```
%DIRECTORIO_TOGETHER%/modules/com/togethersoft/modules
```

La representación física en dicho directorio del módulo será un archivo JAR que puede crearse con las herramientas distribuidas con Together, en el que se incluirá un archivo manifest que será el que lea Together a la hora de cargar el módulo. Existen dos tipos de archivos *manifest*, los *manifest.mf*, los *manifest.def*. Los del tipo manifest.mf son los más extendidos presentando una estructura de texto plano con una serie de atributos que se detallan a continuación:

- **Name**: Nombre del módulo.
- **Time**: Indica a Together el tipo de módulo y por tanto su forma de ejecución. Sus valores, entre otros pueden ser *User* (módulos de usuario), *Startup* (módulos de inicio), *OnDemand* (módulos activables), etc.
- **Main-Class**: Nombre completo (cualificado) de la clase raíz del módulo, que por lo explicado anteriormente debe pertenecer al paquete com.togethersoft.modules o a alguno de los paquetes contenidos en él.
- **Folder**: En el caso de módulos de usuario o activables es la carpeta de la interfaz de Together desde la que se pueden llamar.
- **HelpFile**: Archivo HTML de ayuda que puede consultarse desde la interfaz de Together par obtener ayuda para el módulo en cuestión.

7.1.1.3. El módulo *Together* Cliente.

El módulo desarrollado para el subsistema cliente expuesto en este documento, una vez vistas las características básicas de los módulos de Together ya puede definirse.

Como solución de implementación se han tomado las siguientes decisiones:

- El módulo será de tipo Usuario e Inicio, para permitir tanto al usuario de forma manual como a la herramienta durante su arranque la llamada al módulo que responderá a la funcionalidad del cliente, luego la clase raíz implementará las interfaces *IdeScript* e *IdeStartup*.
- El acceso a su funcionalidad se hará a través de los menús de la propia herramienta Together, en "*Tools/Importar Proyecto Together"*, de forma similar al acceso al módulo desarrollado con anterioridad por el "*PFC Mecanos representando proyectos Together - Inserción automática en el repositorio de GIRO*", a fin de que el acceso a ambas funcionalidades se realice de forma similar para el usuario. Se ha considerado que era muy oportuno que tanto la opción de desarrollo para reutilización como el desarrollo con reutilización sean localizadas por el usuario en una localización semejante, para facilitar su acceso.
- El archivo *manifest* será de tipo *manifest.mf*, puesto que es la opción más difundida y su contenido será:

```
Manifest-Version: 1.0
Name: Importar Mecano
Time: Startup
```

```
Main-Class:
com.togethersoft.modules.ImportarMecanoGIRO.ModuloImportarMecan
o
Created-By: Sanz Martin, Javier

Name: Importar Mecano
Time: User
Main-Class:
com.togethersoft.modules.ImportarMecanoGIRO.ModuloImportarMecan
o
Folder: "Mecano/ImportarMecano"
Created-By: Sanz Martin, Javier
HelpFile:"$TGH$/modules/com/togethersoft/modules/ImportarMecano
GIRO/help/index.html"
```

7.1.1.4. El modelo de Mecano y el Desarrollo con Reutilización con *Together*.

Para la implementación de la importación de elementos reutilizables desde un repositorio basado en el modelo de Mecano es necesario definir la forma en que dichos elementos reutilizables serán representados en Together.

La solución a este problema la proporciona la pregunta*: "¿Qué partes del modelo de Mecano son necesarias para importar assets o mecanos en Together?"* Con lo expuesto en apartados anteriores la respuesta es simple: sólo es necesario conocer la representación física de los assets y su tipología (diagramas, fuentes, etc).

La explicación de esta simplificación a primera vista excesiva del modelo de mecano es la siguiente:

- Para importar un elemento reutilizable, debe obtenerse éste y todos aquellos elementos reutilizables con los que tenga una relación (directa o indirectamente) de carácter fuerte.
- Con el modelo de mecano implementado en el repositorio de GIRO, pueden recorrerse todas las relaciones de dicho modelo desde el acceso al repositorio del servidor.
- A nivel de implementación y con las premisas anteriores, para el cliente situado en la herramienta Together no hay distinción entre un mecano y un asset descargados del repositorio, pues ambos se le presentan como un grupo de assets, la mayoría de los cuales fueron extraídos porque eran necesarios para la reutilización de otro. El cliente deberá importarlos a un proyecto Together sin hacer distinción entre ellos, pues todos son necesarios para la reutilización del asset o mecano descargado.
- Conociendo la tipología de los assets descargados puede especializarse su tratamiento, pues los métodos de inserción en un proyecto Together variarán de un tipo de asset a otro.

Como pudo entreverse en el Documento de Diseño, sólo será necesario especializar el tratamiento de los assets para trasladar el modelo de mecano a un proyecto Together (en importación, no en exportación), recayendo sobre el servidor (puesto que es el único que puede hacerlo) el recorrido de las relaciones semánticas entre los assets del repositorio y su extracción en función de su necesidad para la reutilización.

La especialización del tratamiento de los assets se realizará en función de su tipología:

- **Diagramas**: Su representación física en el repositorio es un documento XMI (archivo XML), que es el modelo de intercambio de datos de Together y otras muchas herramientas. Su tratamiento consistirá en su fusión junto a otros del mismo tipo en un único documento XMI que se importará en Together gracias a la funcionalidad que dicha herramienta proporciona.
- **Códigos Fuente**: Su representación física en el repositorio son archivos de texto plano (extensión .java para los fuentes escritos en JAVA). Su tratamiento requiere la obtención del paquete del proyecto Together al que pertenecen y su inserción el mismo.
- **Protectos Rational**: Su representación física en el repositorio son archivos de texto plano (con extensión MDL) y su tratamiento consistirá en su copia al proyecto de Together, no en su inserción ya que no se tiene seguridad de la fiabilidad de la importación de dichos documentos mediante las funciones de Together.

En los siguientes apartados referidos al módulo de Together Cliente se hace referencia a los componentes necesarios para el tratamiento de los assets diagramas y de los assets códigos fuentes.

7.1.1.5. DOM y los documentos XMI.

Para el tratamiento de los documentos XMI presentados en archivos XML es conveniente su procesado previo para convertirlos en documentos que puedan ser tratados como un árbol.

Previo paso a este procesamiento hay que tener en cuenta la estructura de los documentos XMI almacenados en el repositorio y que responden la formato Unisys XMI Interchange. Este formado de archivo XML sigue la estructura definida en la DTD del módulo de intercambio de datos de Together, ubicada en:

```
%TOGETHER_HOME%/modules/com/togethersoft/modules/interchange/uml.dtd
```

En el repositorio los assets diagramas están almacenados de forma individual, es decir, un asset sólo contiene un diagrama y la estructura que presenta es:

- Una cabecera común a todos (indica formato, herramienta que lo creo, etc)
- Una sección de contenido, donde se especifican los componentes de dicho diagrama sin tener en cuenta su representación)
- Una sección de extensiones, que para el formato Unisys XMI Interchange define la presentación de los elementos de la sección contenido en el diagrama.

Esta estructura determina el algoritmo de tratamiento de estos asset diagramas y que a groso modo (para mayor detalle consultar la documentación de los códigos fuentes que se distribuye en un CD junto a esta memoria) puede describirse así:

- El objetivo es crear un solo documento XMI que englobe a todos los assets diagramas que se desea importar, luego el primer paso es crear la cabecera de dicho documento XMI, y que es común para todos.

- Para cada asset diagrama, se extraerá el la presentación gráfica del diagrama que contiene su documento XMI en la sección de extensiones y se insertará en la sección extensiones del documento XMI que se está creando. Cada elemento de la sección de contenido del asset diagrama referenciado (usado) por la representación del diagrama extraído, será copiado a la sección de contenido del documento XMI común que se está creando.

- Tras procesar todos los diagramas, se ha obtenido un documento XMI común que puede importarse al proyecto Together activo en la herramienta CASE mediante la funcionalidad de que el propio Together dispone.

El tratamiento de estos documentos XMI puede realizarse con un parser XML que responda al modelo DOM (*Document Object Model*). El modelo DOM es una recomendación del W3C (*World Wide Web Consortium*) que consiste en un conjunto de interfaces para diversos lenguajes de programación (para este caso Java) y que permiten el tratamiento de un documento XML a través de los nodos de un árbol de sintaxis abstracta obtenido mediante el parseado del documento XML. La gran ventaja del uso de DOM el la facilidad que presenta para recorrer un documento XML y realizar cambios en él.

El procesamiento del documento XML para su conversión el árbol de sintaxis abstracta se realiza mediante las herramientas conocidas como parsers. Estas herramientas pueden realizar su trabajo comprobando únicamente que el documento este bien formado (parsers sin validación) o verificar además si el documento sigue la estructura definida por una DTD (parsers con validación).

Para el tratamiento de los documentos XMI se ha elegido el parser XML4J que en su versión 2.0.15 (libre distribución) viene incluido con la herramienta Together en su versión 4.2 (sobre la que se desarrollará el sistema que está siendo descrito).

7.1.1.6. Análisis de Código

El problema que plantean los asset códigos fuente es su inserción en el paquete al que pertenecen.

En el caso de que dicho código fuente implemente una clase que se encuentra definida en algún diagrama del proyecto Together sobre el que se va a importar la solución es trivial: se localiza la clase que se corresponda con el nombre del fuente mediante el API de Together, y una vez localizada la clase se obtiene mediante el API de Together a qué paquete pertenece y se inserta el él.

Para el caso en el que el código fuente no implemente a ninguna clase definida en los diagramas de Together la solución no es tan inmediata. La única manera de obtener el paquete al que pertenece la clase contenida en dicho fuente es realizar un

análisis sintáctico del código fuente en base al lenguaje en que fue programado, que para este caso es Java, el lenguaje de todos los asset códigos fuente del repositorio.

Para el análisis de código en Java existe una herramienta de libre distribución en Java llamada JavaCC (*Java Compiler Compiler*) que es un compilador de compiladores. El componente PackageJavaParser es un producto generado por JavaCC a partir de una versión reducida de la sintaxis de Java (que puede consultarse en el CD que acompaña a esta memoria) ofrecida por la propia herramienta. Gracias a este componente, mediante el análisis de código fuente en Java se obtiene el paquete al que dicho fuente pertenece y se inserta en él.

JavaCC a partir de la sintaxis del lenguaje de programación en cuestión, genera una serie de códigos fuente Java, que una vez compilados proporcionarán la funcionalidad de obtención de paquetes deseada.

Estos fuentes aunque no pertenecen al sistema expuesto se distribuirán con él (Ver Documentación de Usuario: Manual de instalación) para su compilación y/o distribución de forma conjunta con este sistema.

7.1.2. *Jakarta-Tomcat* y el subsistema Servidor

Por las directrices del Documento de Requisitos la implementación del servidor se realizará en base a servlets.

La ejecución de un servlet necesita de un motor acoplado a un servidor Web que reciba las peticiones dirigidas al servlet e invoque a los métodos del servlet que las atienden. Una de la mejores apuestas en este sentido es el motor de gestión de Servlets *Jakarta-Tomcat* que aprovecha todas las características adicionales de Java 2.

El proyecto Jakarta es un motor desarrollado para los servidores Web Apache, dándose el caso además de que Apache.org es uno de sus principales promotores y que cuenta con el aliciente de ser gratuito.

Dentro del proyecto Jakarta se ha desarrollado la aplicación Tomcat, que es un contenedor de Servlets bajo un entorno JSP (Java Server Pages) que no es más que una aplicación que maneja e invoca servlets por cuenta del usuario.

El propio servidor en el que se desplegará el software que responde a esta memoria ya cuenta con una distribución de Jakarta-Tomcat, la 3.2 y sobre ella se ejecutará en servlet de esta aplicación.

Como implementación de servlet se ha optado por un servlet HTTP (HttpServlet), que recibirá peticiones HTTP GET o POST. En concreto se ha optado por la opción por defecto de implementación de servlets que es la Multithread (multihilo, puede recibir y atender varias peticiones a la vez). La opción SingleThread limita mucho las aplicaciones que responden a su modelo, pues no permiten atender varias peticiones simultaneas, por lo que fue descartada.

Para poder usar un servlet en Tomcat, se le debe notificar su existencia. Esto se hace gracias a un archivo XML, web.xml, que se define para cada servlet ejecutado por Tomcat (ubicado en directorio WEB-INF de cada aplicación servlet particular) Este archivo proporciona a Tomcat los parámetros de inicialización del servlet, la clase raíz, y lo mas importante, define la dirección URL en la que el servlet (mediante Tomcat) atenderá peticiones. A continuación se describe la estructura del archivo *web.xml* de configuración del servlet de esta aplicación:

```
<web-app>
 <display-name>Universidad de Valladolid</display-name>
 <description>Servlet Importar Mecano</description>
 <context-param>
  <param-name>logFileOut</param-name>
  <param-value>E:\jakarta-tomcat\webapps\ImportarMecano\log\mecano.log</param-
value>
  <description>Fichero log de salida de la aplicacion (Ruta
completa)</description>
  <param-name>logFileErr</param-name>
  <param-value>E:\jakarta-
tomcat\webapps\ImportarMecano\log\mecanoErr.log</param-value>
  <description>Fichero log de salida de error de la aplicacion (Ruta
completa)</description>
 </context-param>
 <servlet>
  <servlet-name>ServletImportar</servlet-name>
  <display-name>Servlet Mecano PFC Julio 2003</display-name>
  <description>PFC E.T.S.I. Informatica Mecanos representando proyectos
Together - Extraccion automatica en el repositorio GIRO</description>
  <servlet-class>ServidorImportar.ServletImportar</servlet-class>
 </servlet>
 <servlet-mapping>
  <servlet-name>ServletImportar</servlet-name>
  <url-pattern>/ServletImportar</url-pattern>
 </servlet-mapping>
</web-app>
```

Los detalles acerca de la instalación del subsistema servidor en Tomcat se darán en la Documentación de Usuario, en el apartado Manual de Instalación

7.1.3. Conexión del subsistema Servidor al Repositorio.

El repositorio al que accederá el servidor se encuentra implementado en una Base de Datos **Oracle 8i**. Para el acceso del subsistema Servidor al repositorio, Java proporciona un API llamado JDBC.

Oracle proporciona diversas soluciones para el aceso a una base de datos a través de JDBC de una aplicación Java, desde drivers JDBC OCI que hacen uso de métodos nativos (lenguaje C) para aumentar el rendimiento (perdiendo portabilidad) , a drivers JDBC Thin, escritos por completo en Java y que suponen la solución que mayor portabilidad ofrece.

Para este caso se ha optado por el driver JDBC Thin, puesto que es el driver descrito en el archivo de configuración común para el acceso a repositorio. En dicho archivo se especifican el protocolo identificación y configuración apropiadas para que una aplicación acceda a la base de datos a través de JDBC. El archivo tiene estructurada esta información en base a atributos (*%atributo%=%valor%*) con lo que puede modificarse el modo de acceso a la base de datos a voluntad. Para este caso

concreto se ha optado por no modificar el archivo de configuración, pues es de uso común a muchas aplicaciones y un cambio podría afectar a su funcionamiento.

El contenido del archivo de configuración es el siguiente:

```
#########################################################################
# Fichero de configuracion del Repositorio GIRO
# Configuracion del Sistema Gestor de Base de Datos
#########################################################################
# Autor: Raul Marticorena Sanchez
# Fecha de creacion: 12/2/99
#########################################################################
# Fecha de ultima modificacion:
# Autor:
# Modificacion:
#########################################################################
#
# USERID
#########################################################################
#  usuario de la base de datos
userid=EXPLOTACION
#
# PASSWORD
#########################################################################
#  password de la base de datos
password=EXPLOTACION
#
# DRIVER
#########################################################################
# driver java para acceder a la base de datos.
# En Oracle 8i tenemos dos opciones:
# a) Driver thin = jdbc:oracle:thin
#       Caracteristicas: mejor portabilidad / menor rendimiento
# b) Driver oci8 = jdbc:oracle:oci8
#       Caracteristicas: menor portabilidad / mayor rendimiento
driver=jdbc:oracle:thin
#
# HOST
#########################################################################
# define el host donde reside la base de datos
host=marte
#
# PROTOCOLO
#########################################################################
# protocolo de acceso a la base de datos
protocol=tcp
#
# PORT
#########################################################################
# puerto por donde se accede a la base de datos
port=1521
#
# SID
#########################################################################
# SID sid de la base de datos
sid=giro
#########################################################################
#########################################################################
#########################################################################
```

En un trabajo previo (*PFC Mecanos representando proyectos Together - Inserción automática en el repositorio de GIRO*) se desarrollo un clase Java que permitiera independientemente de la aplicación que la incluyera el acceso uniforme a la base de datos del repositorio a través del archivo anteriormente descrito. Para este sistema que se está exponiendo se ha optado por el uso de la citada clase (respetando el contenido e información de autor de la misma) para contribuir a uniformizar el acceso de todas las herramientas Java al repositorio de GIRO, potenciado el uso de un acceso probado y fiable frente a una multitud de modos de acceso al repositorio por cada aplicación que lo use.

7.1.3.1. La extracción de elementos reutilizables del repositorio

El repositorio al que accederá el subsistema Servidor responde al modelo de mecano. Esto determina y facilita el modo de extracción de los elementos reutilizables de repositorio.

Como fue detallado en el apartado 7.1.1.4. , el subsistema Cliente no necesita obtener la representación lógica en el modelo de mecano de los elementos reutilizables que solicite, sólo necesita la representación física de los mismos. La responsabilidad de llevar a cabo la extracción de la representación física de los elementos reutilizables navegando a través del modelo de mecano recae sobre el subsistema Servidor.

El principal problema con el que se encuentra el servidor ante una solicitud de descarga de un elemento reutilizable es la obtención de todos aquellos elementos reutilizables que sean necesarios para la reutilización del primero. Para el caso de la descarga de un asset el problema sería la obtención de todos los assets necesarios para la reutilización de asset a descargar y en el caso de la descarga de un Mecano de la obtención de los assets necesarios para la reutilización de cada uno de los assets de los que el mecano se compone.

El hecho de que el repositorio responda al modelo de mecano aporta de inmediato la solución. El modelo de mecano GIRO (presentado en la introducción de esta memoria) requiere de la existencia de relaciones semánticas entre assets, clasificadas por tipo estructural y con información acerca del carácter (fuerte-débil y el sentido del mismo) de dichas relaciones. Por tanto consultando las relaciones entre assets referidas a un asset concreto en el repositorio, se pueden obtener todos aquellos assets que son requeridos para la reutilización del primero dado el carácter de la relación que los une. De nuevo, para los assets obtenidos de esta forma se volvería aplicar la misma búsqueda del carácter de las relaciones que este asset mantiene con otros en el repositorio hasta haber obtenido todos los assets necesarios para la reutilización de elemento reutilizable que se deseaba recuperar del repositorio.

7.1.4. El subsistema de comunicaciones.

Durante el diseño del sistema se fijaron las bases de un protocolo de comunicaciones lo bastante genérico para ser utilizado en futuras aplicaciones y lo bastante potente para cubrir todas las características exigidas a esta aplicación en particular : peticiones al servidor, resultado de búsquedas y descarga de assets del repositorio.

En este nivel de implementación se optó por incluir el paquete de *Comunicaciones* dentro del paquete Cliente, a fin de identificar claramente cual era su origen, esto es :

`com.togethersoft.moduloes.ImportarMecanoGIRO.Comunicaciones`,

pensando en la existencia de otros paquetes *Comunicaciones* que podrían entrar en conflicto con el nombre del paquete *Comunicaciones* de este sistema.

7.1.4.1. La representación lógica de los assets.

Para la definición de la representación lógica de los elementos reutilizables que el cliente solicita al servidor, se usaban documentos XML cuya estructura fijaba una DTD llamada mecano.dtd. Esta DTD fue desarrollada en un proyecto previo para ser el estándar de intercambio de representación lógica de elementos reutilizables entre el repositorio y sus posibles clientes o viceversa. La estructura de dicha DTD es la siguiente:

```
<?xml encoding="UTF-8"?>

<!ELEMENT Mecano Connection (ElementoReutilizable*, Relacion*)>

<!ELEMENT ElementoReutilizable (Asset | Mecano)>

<!ELEMENT Asset (nombre, resumen?, fechaCreacion?, fechaIntroduccion,
   fechaModificacion?, palabrasClave?, plataforma?,
   restricciones?, nivelSeguridad?, coste?, idioma?,
   version?, nivelEvaluacion?, autores?, palabras?,
   tipoasset, metodo?, nivelAbstraccion, Representacion?)>

<!ELEMENT Representacion (nombre, URL, formato, tamanno, medio, comentarios?)>

<!ELEMENT Mecano (nombre, resumen?, fechaCreacion?, fechaIntroduccion,
   fechaModificacion?, palabrasClave?, plataforma?,
   restricciones?, nivelSeguridad, coste?, idioma?,
   version?, nivelEvaluacion?, autores?, palabras?,
   paradigma?)>

<!ELEMENT Relacion (nombre, idAssOrigen, idAssDestino, tipoRelacion)>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT resumen (#PCDATA)>
<!ELEMENT fechaCreacion (#PCDATA)>
<!ELEMENT fechaIntroduccion (#PCDATA)>
<!ELEMENT fechaModificacion (#PCDATA)>
<!ELEMENT palabrasClave (#PCDATA)>
<!ELEMENT plataforma (#PCDATA)>
<!ELEMENT restricciones (#PCDATA)>
<!ELEMENT nivelSeguridad (#PCDATA)>
<!ELEMENT coste (#PCDATA)>
<!ELEMENT idioma (#PCDATA)>
<!ELEMENT version (#PCDATA)>
<!ELEMENT nivelEvaluacion (#PCDATA)>
<!ELEMENT autores (#PCDATA)>
<!ELEMENT palabras (#PCDATA)>
<!ELEMENT tipoasset (#PCDATA)>
<!ELEMENT metodo (#PCDATA)>
<!ELEMENT nivelAbstraccion (#PCDATA)>
<!ELEMENT paradigma (#PCDATA)>
<!ELEMENT idAssOrigen (#PCDATA)>
<!ELEMENT idAssDestino (#PCDATA)>
<!ELEMENT tipoRelacion (#PCDATA)>
<!ELEMENT URL (#PCDATA)>
<!ELEMENT formato (#PCDATA)>
<!ELEMENT tamanno (#PCDATA)>
<!ELEMENT medio (#PCDATA)>
<!ELEMENT comentarios (#PCDATA)>
```

Esta DTD da soporte tanto a assets como a mecanos y responde al modelo de mecano GIRO expuesto en la Introducción y además proporciona información relevante acerca del elemento reutilizable cuya representación física se encuentren un documento XML que responda a esta DTD.

Con el uso de documentos XML basados en esta DTD el servidor puede mandar información muy precisa al cliente. En particular puede mandar esta información como resultado de la petición del cliente de la búsqueda de los elementos reutilizables que se

ajusten a un determinado patrón. La representación lógica de un asset sería mantenida en un documento XML por la clase definida por el código fuente *ItemResultado.java*.

El problema podría surgir ante el procesado (parseado) simultáneo de todos los resultados obtenidos en la búsqueda por parte del cliente, ya que el rendimiento se resentiría. Por eso, en el nivel de implementación del sistema, la representación lógica de un asset sólo es parseada cuando se la necesita (cuando se desea obtener la información relevante de un resultado de la búsqueda por parte de cliente) y sólo es procesada una vez, manteniéndose la información parseada en una estructura que agilice su consulta. Para este caso en particular se ha optado por una tabla Hash accediéndose a su contenido por el nombre de la característica o propiedad definida en la DTD que se desea consultar (nombre, versión, autores, etc)

7.1.4.2. La privacidad de los datos del usuario.

Desde el apartado de análisis se dejo claro que para asegurar la privacidad de la identificación del usuario de la herramienta CASE al utilizar este sistema que se esta describiendo, dicha identificación (*login* y *password*) deberían encriptarse.

Como solución se optó por los sistemas de encriptado basados en clave pública, puesto que son el método más oportuno para comunicaciones de cliente-servidor en un sistema distribuido de múltiples clientes. La clave pública de encriptado sería común y conocida por todos los clientes, con la que encriptarían sus identificaciones, que solo podrían ser leídas por el servidor que es el único que posee la clave privada con la que desencriptar la identificación. Además este sistema de criptografía permite al servidor firmar con su clave privada la clave pública que distribuye a sus clientes para asegurarles su autenticidad.

Sin embargo se a pospuesto hasta esta etapa de implementación la elección de algoritmo de encriptado basado en clave pública que se empleará.

Para el uso de criptografía el lenguaje Java 2 proporciona una extensión consistente en una serie de interfaces en el paquete *javax.crypto*, que se distribuye de forma conjunta con Java a partir de la versión 1.4.1 aunque está disponible para versiones anteriores. Esta extensión es conocida como JCE (*Java Cryptography Extensions*) y actualmente se encuentra en su versión 1.2.2.

Como se explicó en el párrafo anterior JCE sólo proporciona una serie de interfaces para el uso de diversos algoritmos de encriptación con Java (*DES*, *IDEA*, *RSA*, etc), pero debido a la existencia de patentes en muchos de los algoritmos de encriptación (por ejemplo IDEA para uso comercial), la implementación de dichos algoritmos ha de obtenerse aparte. Nace así el concepto de Proveedor de Seguridad para Java (*Security Provider*) que es usado también para otras características de Java 2 no relacionadas con la criptografía.

Existen unos pocos proveedores para JCE en todo el mundo. Algunos son gratuitos (*GNUCrypto*, *BouncyCastle*, *SunJCE*) y otros no (*RSA Laboratories*). No todos los proveedores de JCE soportan algoritmos de encriptación basados en clave

pública (*SunJCE*) o aún esta implementación de algoritmos de clave pública está en desarrollo (*GNUCrypto*).

Para este sistema en concreto e independientemente del algoritmo de encriptación usado se ha optado por el proveedor BouncyCastle (*Legion of the Bouncy Castle*, `http://www.bouncycastle.org`) que es un proveedor de JCE gratuito y de código fuente abierto, tanto para uso comercial como personal cuyo uso esta regulado por una licencia basada en el *MIT/X Consortium*.

Además, a esto hay que añadir las siguientes ventajas:

- Alto rendimiento para tratarse de criptografía desarrollada 100% en Java.
- Gran cantidad de algoritmos implementados y soporte para casi todos los estándares de encriptado (ANSI X509, ANSI X9, ISO/IEC).
- Soporte de JCE para clientes ligeros, lo que facilita su utilización en sistemas móviles de capacidad de almacenamiento reducida (PDA's, telefonía móvil con soporte para Java, etc).
- Distribuciones de JCE y del proveedor de JCE para cualquier versión de Java 2 (JDK 1.2.2, JDK 1.3.1, JDK 1.4.1)
- Código fuente abierto a consulta y modificación.

Como inconvenientes a este proveedor (y al resto de proveedores) es la existencia de algoritmos patentados por los que se ha de pagar una cantidad o royalties, o que no permiten su utilización para fines comerciales (como es el caso de IDEA).

Dado que tanto el cliente como el servidor de este sistema que se esta detallando será ejecutados en una JVM 1.3.1, se ha optado por la distribución para JVM 1.3.1 del proveedor de JCE BouncyCastle, que actualmente se encuentra por la versión 1.18.

Una vez escogido el proveedor de criptografía para Java, queda la elección del algoritmo de encriptado basado en clave pública. No hay gran variedad de estos algoritmos que hayan sido implementados en proveedores de JCE (ni tan siquiera en el proveedor BouncyCastle u otros de pago como el de RSA Laboratories) y de éstos algunos están sometidos a patente.

Aprovechando que el algoritmo RSA (*Rivest-Shamir-Adleman*, 1977) fue el primero en aparecer y hoy en día aún es referencia por su robustez y fiabilidad, se ha optado por este algoritmo de encriptado basado en clave pública. El algoritmo RSA estaba patentado a nombre del Massachussets Institute of Technology (patente *#4405829*), pero la patente caducó en el 2000 (las patentes de estos algoritmos tienen una validez de entre 17 y 20 años, ver *APENDICES* para mayor información).

Por tanto se empleará el algoritmo RSA para que los clientes encripten su identificación y sólo el servidor pueda leerla, y también se empleará para que el servidor firme la clave pública que distribuye a los clientes, a fin de que estos comprueben su autenticidad.

7.2 Diagrama de Despliegue

El software desarrollado necesita desplegarse sobre el hardware para su ejecución. A este nivel es necesario considerara la ubicación de los componentes sobre el hardware existente. UML proporciona para representar esta tarea los diagramas de despliegue en el que se muestran los nodos sobre los que se desplegarán los componentes sin especificar un hardware concreto.

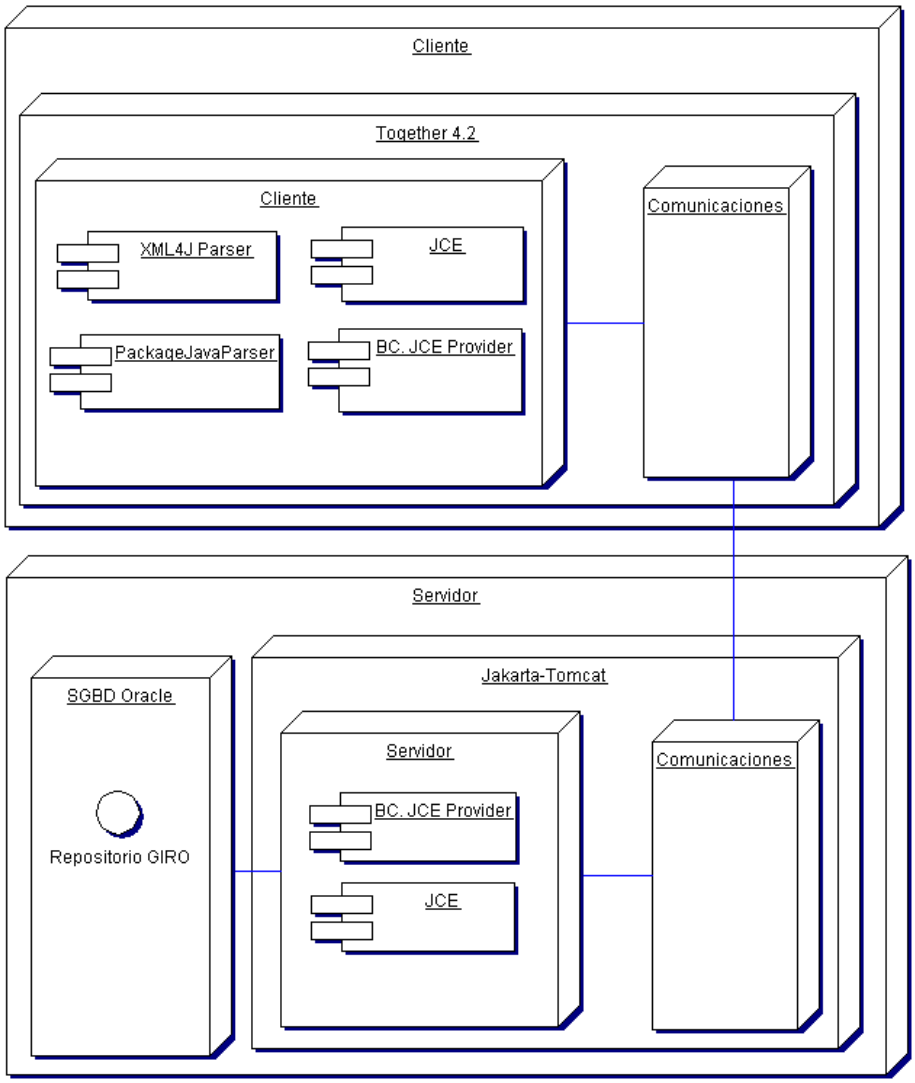

Figura 7.2. Diagrama de Despliegue

En este diagrama se observa que el sistema estará compuesto de un cliente (que pueden ser varios) y un servidor, los cuales se comunican a través de un nodo común de Comunicaciones que tiene desplegada una copia en cada uno (una para el cliente y otra para el servidor).

8. DOCUMENTACIÓN DE USUARIO

8.1. Manual de Usuario

Este documento pretende ser una guía de uso y manejo del sistema de cara al usuario. Dado que el sistema sólo interactúa con el usuario en el cliente, embebido en la herramienta CASE Together 4.2, todo este apartado tratará de explicar al usuario como utilizar la funcionalidad del sistema.

La utilización del sistema es muy fácil pero aún así se detallarán todas las opciones que el usuario puede realizar de forma detallada.

8.1.1. Acceso a las funcionalidades del sistema

La funcionalidad del sistema se encuentra disponible a través del menú de la herramienta Together 4.2. En concreto, se accederá a estas funcionalidades desde el menú Tools y el submenú Importar Proyecto Together.

En la siguiente figura se puede observar los menús indicados anteriormente y las opciones que proporcionan.

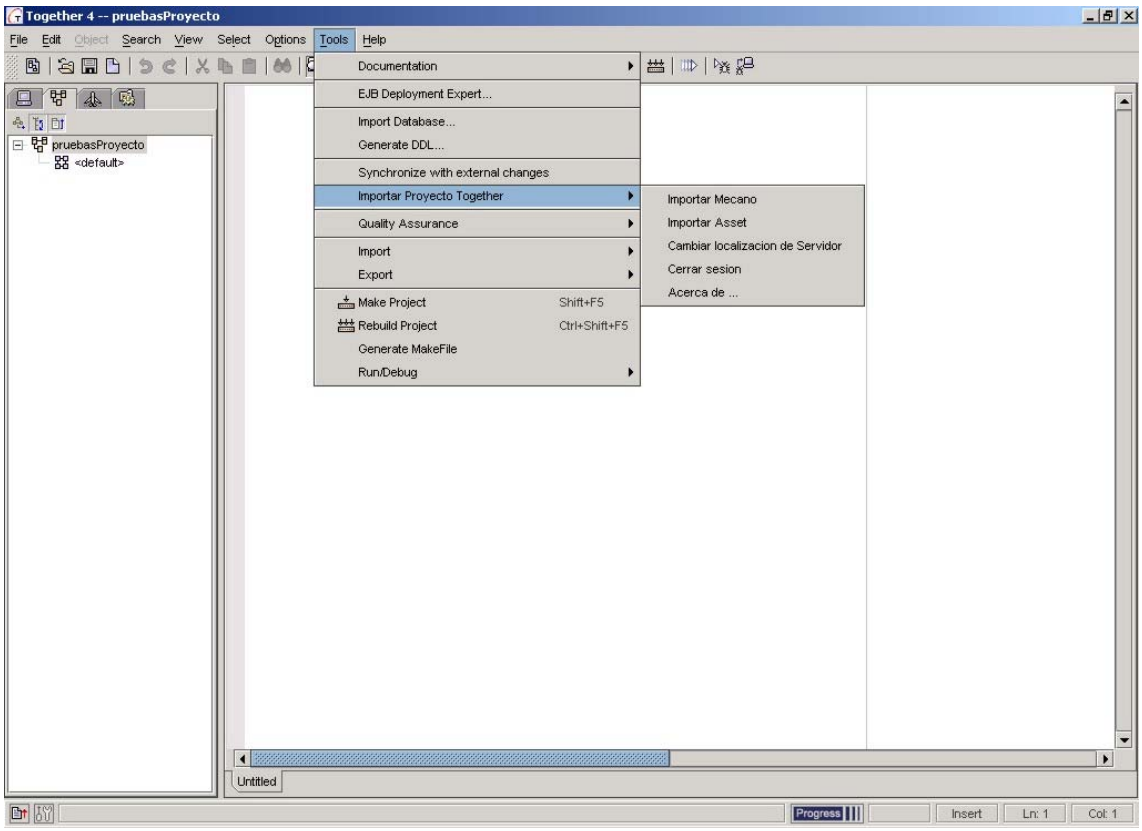

Figura 8.1. Acceso a la funcionalidad del sistema.

8.1.2. Elección de funciones

A continuación se describen todas y cada una de las funciones que puede llevar a cabo el sistema, así como la manera de realizarlas.

8.1.2.1. Importar Mecano.

Con esta opción el usuario puede importar sobre el proyecto Together en que se encuentre trabajando un Mecano del repositorio software de GIRO.

Se accede a esta función desde el menú "*Tools/Importar Proyecto Together*" de la herramienta CASE y seleccionado después la opción de menú "*Importar Mecano*".

Durante todo el proceso la página "*Importar Mecano*" de la ventana de mensajes de Together informará al usuario mediante mensajes que es lo que ocurre en el sistema.

Es necesario que el usuario tenga un proyecto abierto sobre el que importar el Mecano. De lo contrario el sistema cancelará la operación indicándole al usuario el motivo (*"No hay ningún proyecto abierto. Abra o cree uno nuevo"*).

1.- El primer paso que el usuario debe realizar tras elegir esta opción es identificarse. El sistema le pedirá un login y un password para confirmar su identidad y tras haberlos introducido hará Click con el ratón en el botón OK.

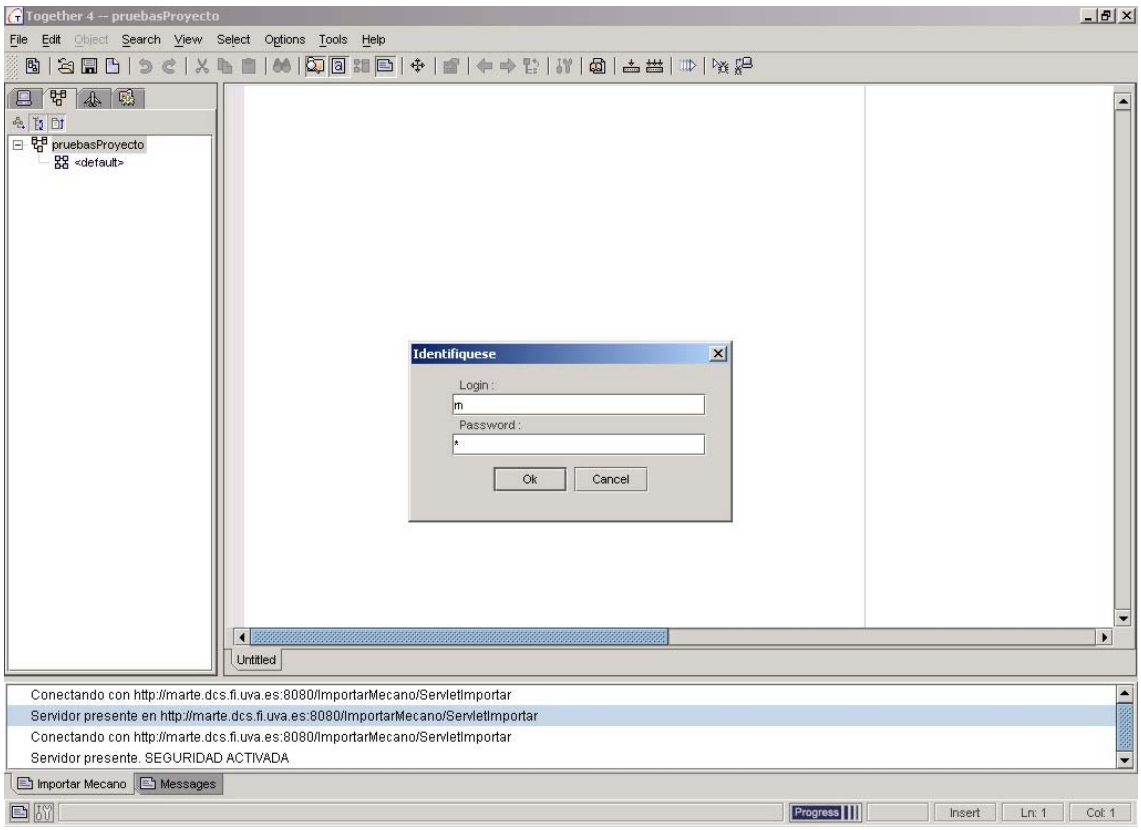

Figura 8.2. Identificación de usuario

Para facilitar el uso del sistema el usuario sólo deberá identificarse una vez pues se permite el trabajo en sesiones. Después de una identificación correcta el sistema recordará quién es el usuario, y éste siempre dispondrá de la opción de cerrar su sesión y que su identificación desparezca del sistema (función 8.1.2.4 Cerrar sesión)

Si la identificación del usuario es incorrecta, o éste cancela la operación (botón CANCEL de la ventana de identificación) o no se puede establecer comunicación con el servidor, se cancelará la operación informando al usuario del motivo de la cancelación.

2.- Tras haberse identificado correctamente, o llamar a esta función del sistema desde una sesión de trabajo abierta (identificación afirmativa con anterioridad), se mostrará al usuario una ventana para que indique el patrón de búsqueda al que debe ajustarse el Mecano que desea importar a su proyecto en Together.

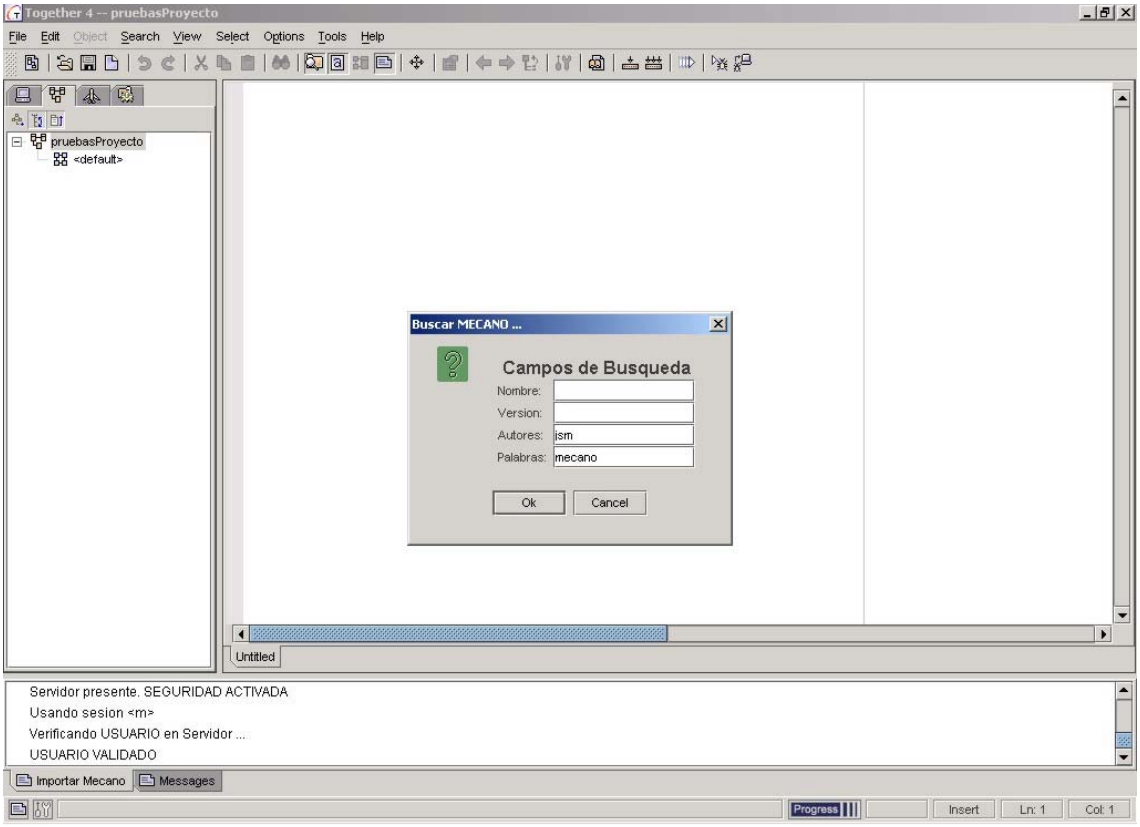

Figura 8.3. Selección del patrón de búsqueda de Mecano.

Los campos de búsqueda serán nombre, versión, autores y palabras (palabras clave que definen el elemento a buscar). Tras haberlos introducido el usuario hará Click con el ratón en el botón OK.

Se buscará en el repositorio los Mecanos que CONTENGAN en las propiedades anteriormente nombradas EXACTAMENTE el valor introducido por el usuario sin hacer distinción entre mayúsculas y minúsculas, a no ser que en ese campo de búsqueda el usuario no haya introducido ningún valor.

En el caso de que el usuario no introduzca ningún valor en ningún campo buscarán todos los Mecanos del repositorio, pero antes se solicitará la confirmación de usuario para realizar esta búsqueda ya que puede llevar algún tiempo.

Ante la cancelación de la búsqueda por parte del usuario, o un error en la comunicación con el servidor se cancelará la operación.

En el caso de que no se encontrara ningún Mecano en el repositorio que se ajuste al patrón de búsqueda indicado por el usuario, se mostraría una ventana de error al usuario informándole de dicha eventualidad y cancelará la operación.

3.- Si el sistema encontró en el repositorio algún (o algunos) Mecano que se ajustara al patrón de búsqueda indicado por el usuario, se los presentará en forma de listado al usuario para un examen más detallado.

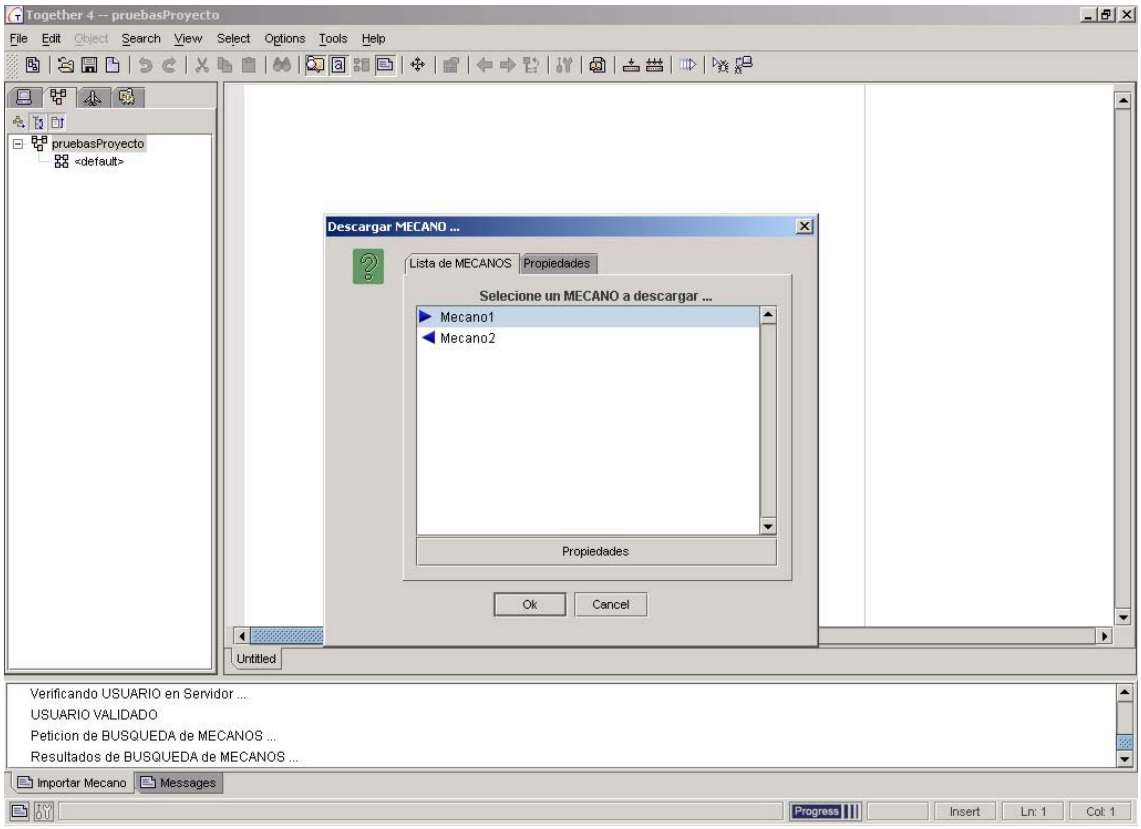

Figura 8.4. Listado de Mecanos encontrados.

En este punto el usuario debe seleccionar algún Mecano para su importación en el proyecto Together activo. Ante la selección de un Mecano de la lista por el usuario, este se mostrará resaltado sobre el resto (cambiando de orientación un pequeño icono a su izquierda) como puede observarse en la figura 8.4.

Se permite al usuario cambiar de selección y seleccionar otro Mecano.

También se permite que el usuario observe las características del Mecano de forma detallada haciendo Click con el ratón en el botón "*Propiedades*" o en la pestaña

del mismo nombre. Le aparecerá entonces la información detallada del mecano que tenía seleccionado.

Si una de las informaciones es de un texto tan largo que no puede presentarse en la ventana de información, el usuario podrá verla ampliada colocando el cursor del ratón sobre esa información como se observa en la siguiente figura.

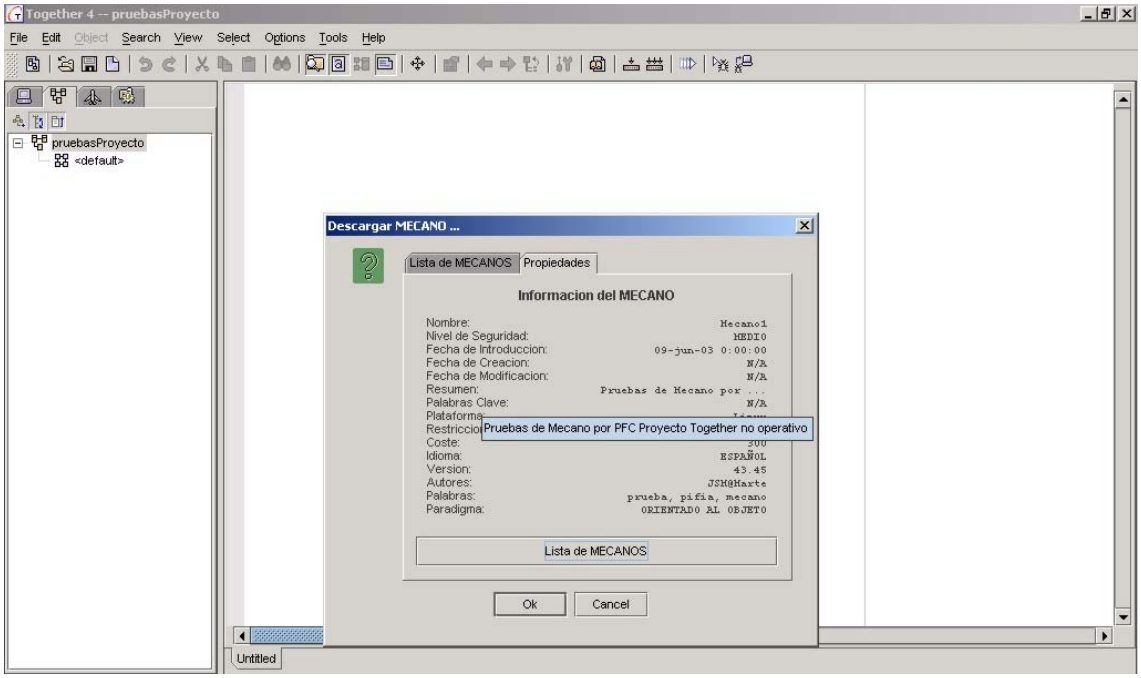

Figura 8.5. Propiedades de un Mecano.

El usuario puede volver a la ventana de selección de Mecanos haciendo Click con el ratón sobre el botón "*Lista de MECANOS*" o la pestaña del mismo nombre.

Cuando el usuario haga Click con el ratón sobre el botón OK en cualquiera de las dos ventanas comenzará la descarga del Mecano desde el repositorio.

Si el usuario hiciera Click sobre el botón CANCEL o no hubiera ningún Mecano seleccionado en la lista al presionar el botón OK se cancela la operación y se informa de ello al usuario.

4.- El servidor envía al cliente el Mecano solicitado y este lo importa sobre el proyecto Together abierto.

Ante un error en la comunicación con el servidor o una inconsistencia de la base de datos que impida la descarga de todos y cada uno de los Assets del Mecano que mantiene el repositorio se cancela la operación comunicándole tal eventualidad al usuario. Los errores puntuales por inconsistencia de la base de datos no supondrán la cancelación de la operación y su tratamiento será transparente al usuario.

Descargado el Mecano, Together solicitará al usuario la selección de un archivo en una ventana de selección de archivo que contiene le proyecto (en este caso Mecano) que se va a importar. Previamente el sistema ha informado al usuario de que archivo se

trata. Es el archivo *ProyectoXMI.xml* situado en el directorio raíz del proyecto activo en Together. Su localización no puede ser más simple.

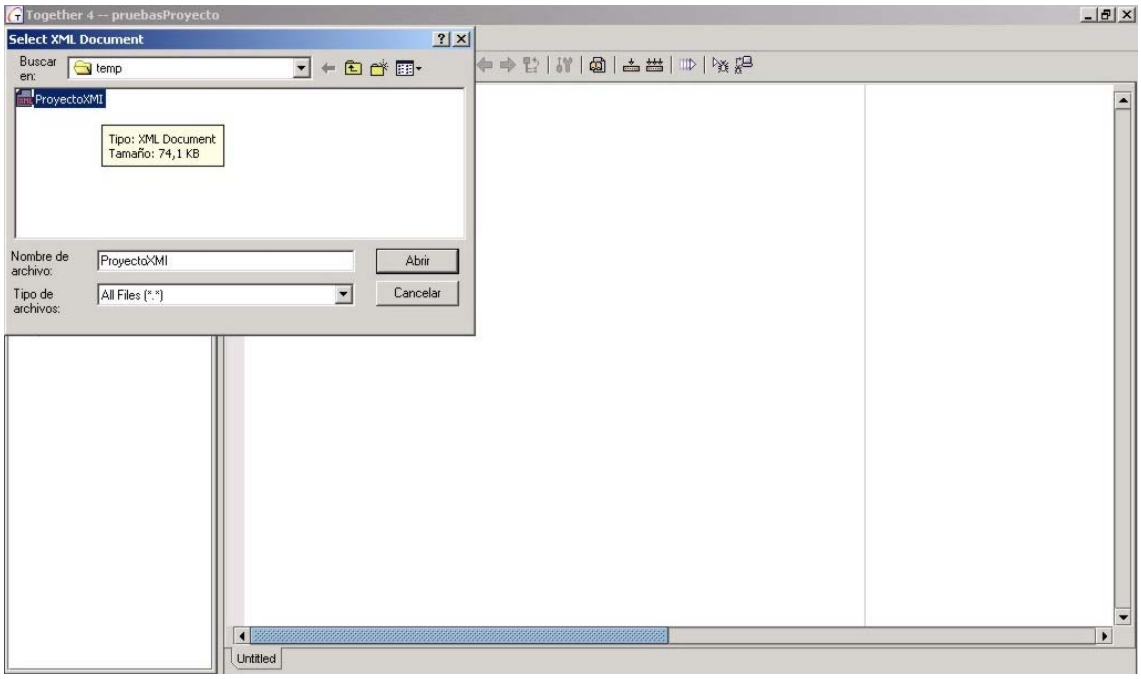

Figura 8.6. Selección del proyecto a importar.

Durante el proceso de importación del Mecano el sistema informará desde la página "Importar Mecano" de la ventana de mensajes de Together que labor está realizando, así como los eventuales mensajes de error que puedan producirse. Estos errores puntuales de importación no supondrán la cancelación de la operación.

Tras completarse la importación del Mecano, se actualizará el entorno de trabajo de Together con los elementos importados y ene la página "*Importar Mecano*" de la ventana de mensajes de Together aparecerá el texto " *PROYECTO IMPORTADO CON ÉXITO* ".

Lo que sucede siempre en el servidor se visualizarán en dos ficheros estándar de salida, que se localizan en `%DIRECTORIO_TOMCAT%/webapps/ImportarMecano/logs`. En este directorio encontraremos la salida estándar, *mecanoOut.log*, y la salida de error, *mecanoErr.log*.

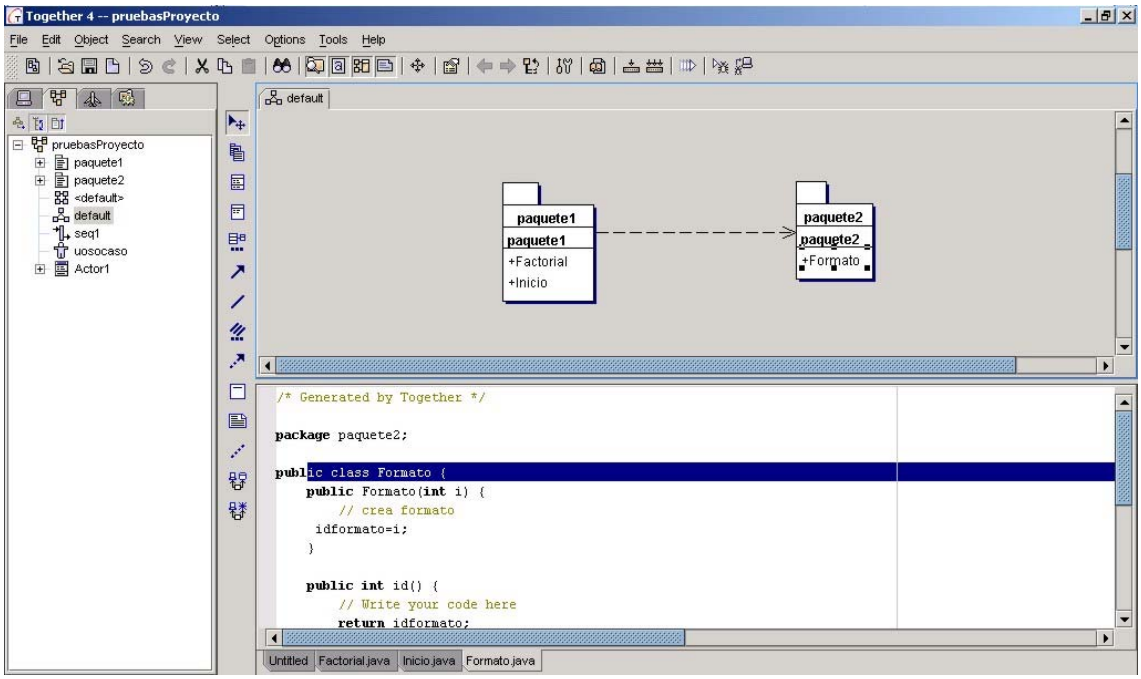

Figura 8.7. Fin de la importación de un Mecano

8.1.2.2. Importar Asset.

Con esta opción el usuario puede importar sobre el proyecto Together en que se encuentre trabajando un Asset del repositorio software de GIRO. Su funcionamiento es idéntico totalmente al funcionamiento de la función de "*Importar Mecano*", pero para aclarar posibles dudas al usuario acerca de su funcionamiento será detallado a continuación.

Se accede a esta función desde el menú "*Tools/Importar Proyecto Together*" de la herramienta CASE y seleccionado después la opción de menú "*Importar Asset*".

Durante todo el proceso la página "*Importar Mecano*" de la ventana de mensajes de Together informará al usuario mediante mensajes que es lo que ocurre en el sistema.

Es necesario que el usuario tenga un proyecto abierto sobre el que importar el Asset. De lo contrario el sistema cancelará la operación indicándole al usuario el motivo ("*No hay ningún proyecto abierto. Abra o cree uno nuevo*").

1.- El primer paso que el usuario debe realizar tras elegir esta opción es identificarse. El sistema le pedirá un login y un password para confirmar su identidad y tras haberlos introducido hará Click con el ratón en el botón OK. La ventana que mostrará el sistema es la misma que se presentó en la figura 8.1.

Para facilitar el uso del sistema el usuario sólo deberá identificarse una vez pues se permite el trabajo en sesiones. Después de una identificación correcta el sistema recordará quién es el usuario, y éste siempre dispondrá de la opción de cerrar su sesión y que su identificación desparezca del sistema (función 8.1.2.4 Cerrar sesión)

Si la identificación del usuario es incorrecta, o éste cancela la operación (botón CANCEL de la ventana de identificación) o no se puede establecer comunicación con el servidor, se cancelará la operación informando al usuario del motivo de la cancelación.

2.- Tras haberse identificado correctamente, o llamar a esta función del sistema desde una sesión de trabajo abierta (identificación afirmativa con anterioridad), se mostrará al usuario una ventana para que indique el patrón de búsqueda al que debe ajustarse el Asset que desea importar a su proyecto en Together.

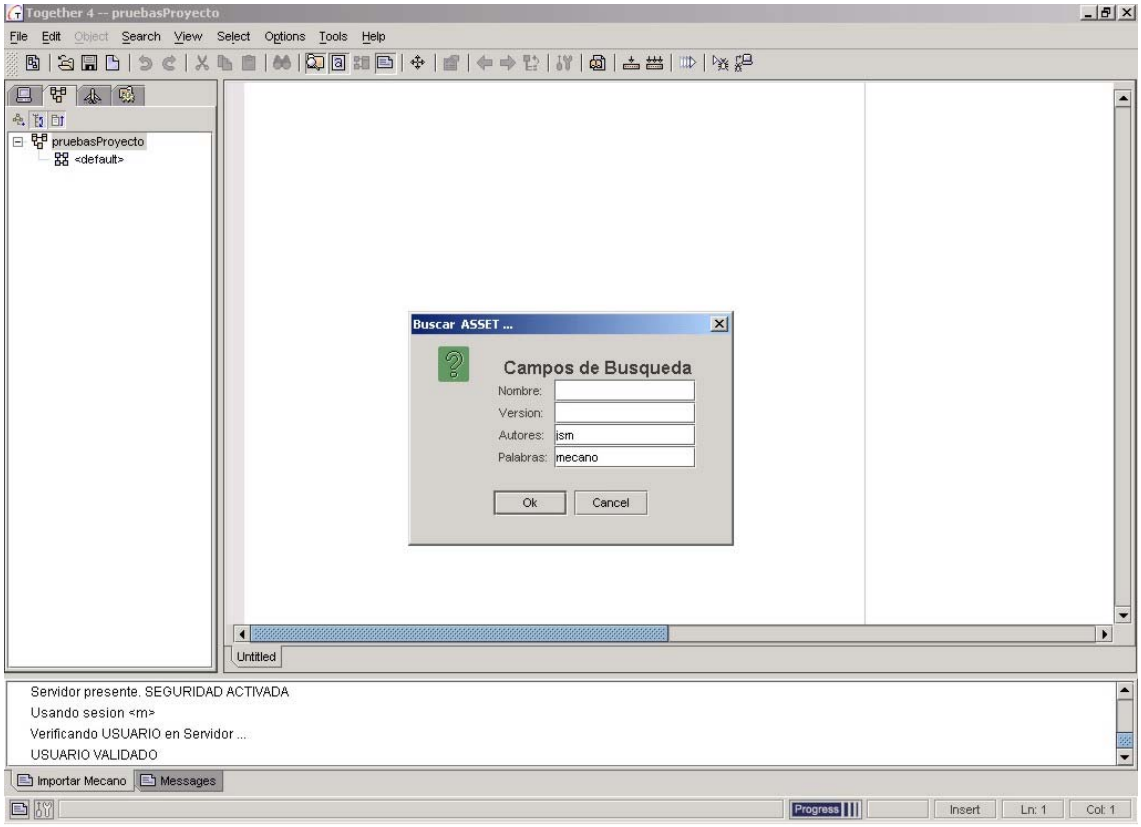

Figura 8.8. Selección del patrón de búsqueda de Asset.

Los campos de búsqueda serán nombre, versión, autores y palabras (palabras clave que definen el elemento a buscar). Tras haberlos introducido el usuario hará Click con el ratón en el botón OK.

Se buscará en el repositorio los Assets que CONTENGAN en las propiedades anteriormente nombradas EXACTAMENTE el valor introducido por el usuario sin hacer distinción entre mayúsculas y minúsculas, a no ser que en ese campo de búsqueda el usuario no haya introducido ningún valor.

En el caso de que el usuario no introduzca ningún valor en ningún campo buscarán todos los Assets del repositorio, pero antes se solicitará la confirmación de usuario para realizar esta búsqueda ya que puede llevar algún tiempo.

Ante la cancelación de la búsqueda por parte del usuario, o un error en la comunicación con el servidor se cancelará la operación.

En el caso de que no se encontrara ningún Mecano en el repositorio que se ajuste al patrón de búsqueda indicado por el usuario, se mostraría una ventana de error al usuario informándole de dicha eventualidad y cancelará la operación.

3.- Si el sistema encontró en el repositorio algún (o algunos) Asset que se ajustara al patrón de búsqueda indicado por el usuario, se los presentará en forma de listado al usuario para un examen más detallado.

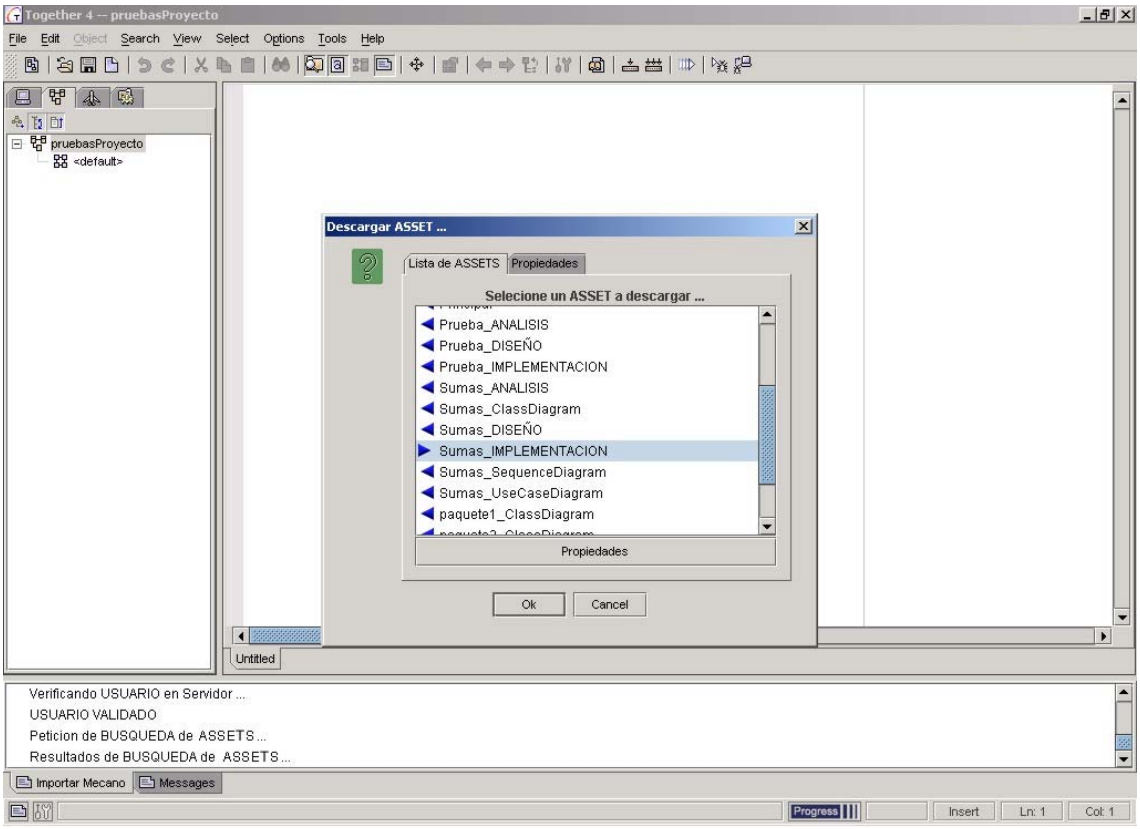

Figura 8.9. Listado de Assets encontrados.

En este punto el usuario debe seleccionar algún Asset para su importación en el proyecto Together activo. Ante la selección de un Asset de la lista por el usuario, este se mostrará resaltado sobre el resto (cambiando de orientación un pequeño icono a su izquierda) como puede observarse en la figura 8.9.

Se permite al usuario cambiar de selección y seleccionar otro Asset.

También se permite que el usuario observe las características del Asset de forma detallada haciendo Click con el ratón en el botón "*Propiedades*" o en la pestaña del mismo nombre. Le aparecerá entonces la información detallada del mecano que tenía seleccionado.

Si una de las informaciones es de un texto tan largo que no puede presentarse en la ventana de información, el usuario podrá verla ampliada colocando el cursor del ratón sobre esa información como se observa en la siguiente figura.

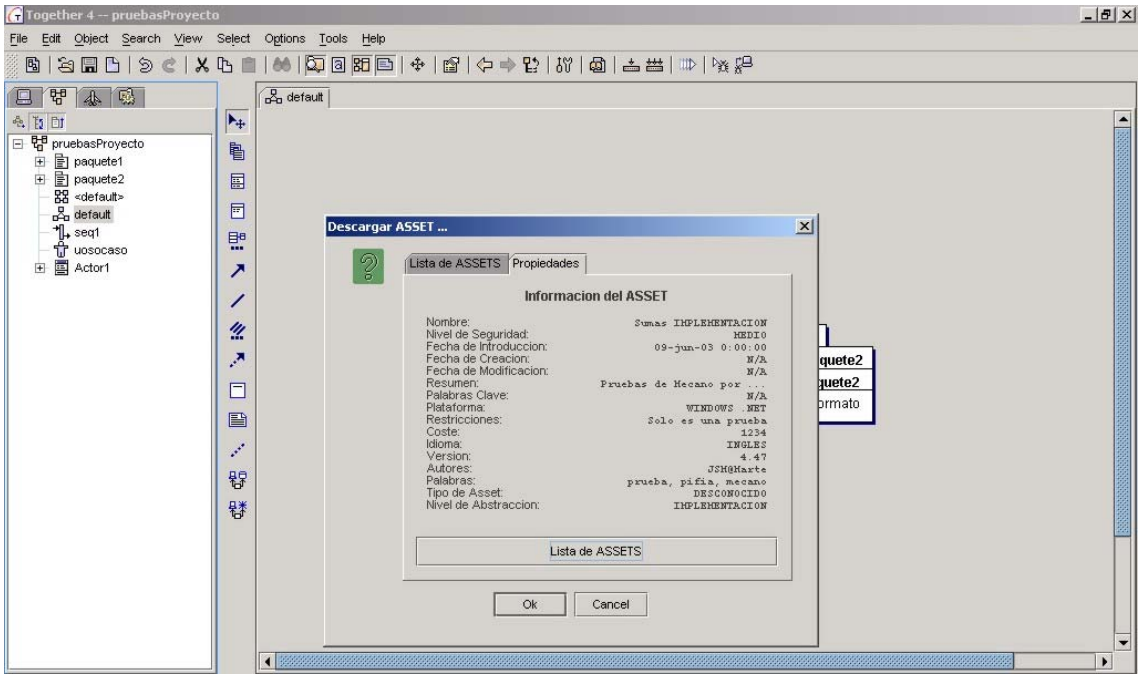

Figura 8.10. Propiedades de un Asset.

El usuario puede volver a la ventana de selección de Assets haciendo Click con el ratón sobre el botón "*Lista de ASSETS*" o la pestaña del mismo nombre.

Cuando el usuario haga Click con el ratón sobre el botón OK en cualquiera de las dos ventanas comenzará la descarga del Asset desde el repositorio.

Si el usuario hiciera Click sobre el botón CANCEL o no hubiera ningún Asset seleccionado en la lista al presionar el botón OK se cancela la operación y se informa de ello al usuario.

4.- El servidor envía al cliente el Asset solicitado y este lo importa sobre el proyecto Together abierto.

Ante un error en la comunicación con el servidor o una inconsistencia de la base de datos que impida la descarga de todos y cada uno de los Assets necesarios para la reutilización del Asset que se pretende descargar (incluido él mismo) se cancela la operación comunicándole tal eventualidad al usuario. Los errores puntuales por inconsistencia de la base de datos no supondrán la cancelación de la operación y su tratamiento será transparente al usuario.

Descargado el Asset, Together solicitará al usuario la selección de un archivo en una ventana de selección de archivo que contiene le proyecto (en este caso Asset) que se va a importar. Previamente el sistema ha informado al usuario de que archivo se trata. Es el archivo *ProyectoXMI.xml* situado en el directorio raíz del proyecto activo en Together. Su localización no puede ser más simple. La ventana de solicitud antes nombrada puede observarse en la figura 8.6.

Durante el proceso de importación del Asset el sistema informará desde la página "Importar Mecano" de la ventana de mensajes de Together que labor está realizando, así como los eventuales mensajes de error que puedan producirse.

Tras completarse la importación del Asset, se actualizará el entorno de trabajo de Together con los elementos importados y ene la página "*Importar Mecano*" de la ventana de mensajes de Together aparecerá el texto " *PROYECTO IMPORTADO CON ÉXITO* ". Es resultado es el mismo que se observaba en la figura 8.7.

Lo que sucede siempre en el servidor se visualizarán en dos ficheros estándar de salida, que se localizan en `%DIRECTORIO_TOMCAT%/webapps/ImportarMecano/logs`. En este directorio encontraremos la salida estándar, *mecanoOut.log*, y la salida de error, *mecanoErr.log*.

8.1.2.3. Cambiar localización de servidor

Con esta opción el usuario puede cambiar la dirección URL a la que el cliente dirigirá las peticiones para el servidor del sistema. Esta personalización permite al sistema, no sólo ser un sistema distribuido de múltiples clientes, sino que además permite la conexión a diversos servidores de repositorio mediante el cambio de la dirección URL de servidor a la que el cliente se conecta.

Se accede a esta función desde el menú "*Tools/Importar Proyecto Together*" de la herramienta CASE y seleccionado después la opción de menú "*Cambiar localización de Servidor*".

Tras la selección de esta función, el sistema mostrará al usuario una ventana solicitándole la nueva dirección URL del servidor, como puede observarse en la figura 8.11., situando por defecto en el nuevo valor al antigua dirección URL de servidor.

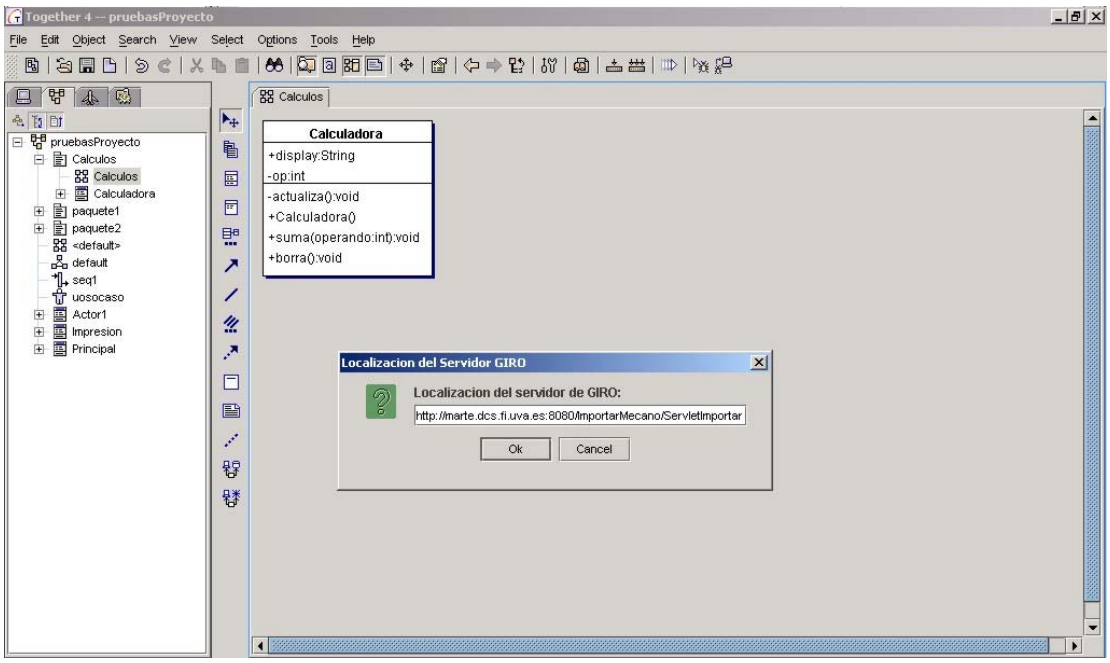

Figura 8.11. Nueva dirección del servidor.

El usuario debe entonces introducir la nueva dirección URL del servidor y hacer Click con el ratón sobre el botón OK. Entonces en cliente efectuará una prueba de conexión con el servidor para comprobar su presencia. El resultado de esa prueba se mostrará al usuario indicando si fue correcta o si se produjeron errores, así como la tipología de los mismos.

Si el usuario cancela la operación pulsando el botón CANCEL, el cliente realizará una prueba de conexión con la antigua dirección URL del servidor informando como fue explicado anteriormente el resultado de dicha prueba de conexión.

8.1.2.4. Cerrar sesión

Con esta función se permite al usuario que su identificación personal (login y password) en el sistema sea borrada. Con esta acción, ante una nueva utilización de las funciones 8.1.2.1 y 8.1.2.1. (Importar Mecano y Asset) se le volverá a solicitad e identificación.

El objetivo de esta función es permitir al usuario el uso de varias cuentas (suponiendo que un usuario poseyera varias identificaciones administrativas en el repositorio) durante su trabajo en Together.

En cualquier caso, la privacidad de la identificación de usuario ante el cierre de Together está garantizado, pues la identificación de usuario que el módulo de Together mantiene desaparecería con él al cerrarse Together (y por tanto finalizar la ejecución del módulo).

Se accede a esta función desde el menú "*Tools/Importar Proyecto Together*" de la herramienta CASE y seleccionado después la opción de menú "*Cerrar sesión*".

Al seleccionar esta función, si no hay ninguna sesión abierta en el sistema, la operación se cancelará informando al usuario, pues no hay ninguna sesión que cerrar.

Si existiera un sesión abierta en el sistema, se solicitará la confirmación del usuario antes de cerrarla, tal y como puede apreciarse en la figura 8.12.

Si el usuario confirma el cierre de sesión su identificación desaparecerá del sistema quedando su sesión de trabajo con el servidor concluida, informándose al usuario de tal eventualidad. A partir de entonces el usuario deberá volver a identificarse si desea acceder al as funcionalidades restringidas del sistema.

Si por el contrario el usuario no confirma el cierre de sesión, se cancela la operación informando al usuario de ello. La sesión del usuario en este sistema aún continúa activa.

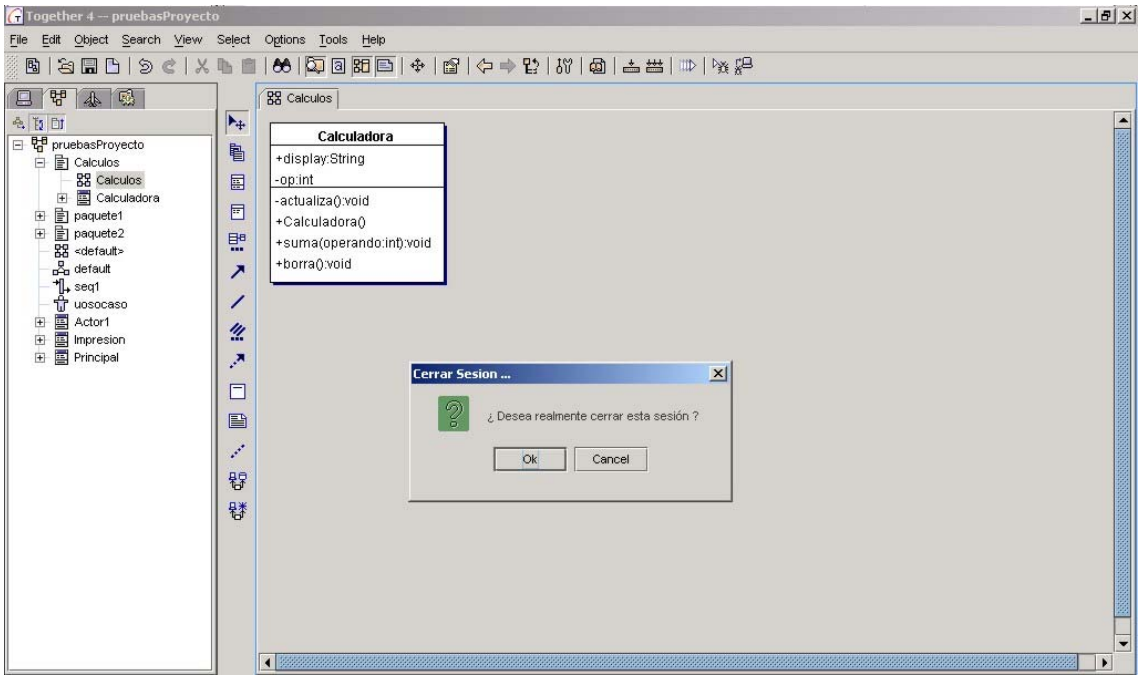

Figura 8.12. Cerrar sesión.

8.1.2.5. Acerca de ...

Esta función sólo presente dar una información al usuario del nombre del sistema así como de su creador.

Se accede a esta función desde el menú "*Tools/Importar Proyecto Together*" de la herramienta CASE y seleccionado después la opción de menú "*Acerca de ...*".

El la siguiente figura se muestra esta información, que se cerrará al pulsar el usuario el botón OK.

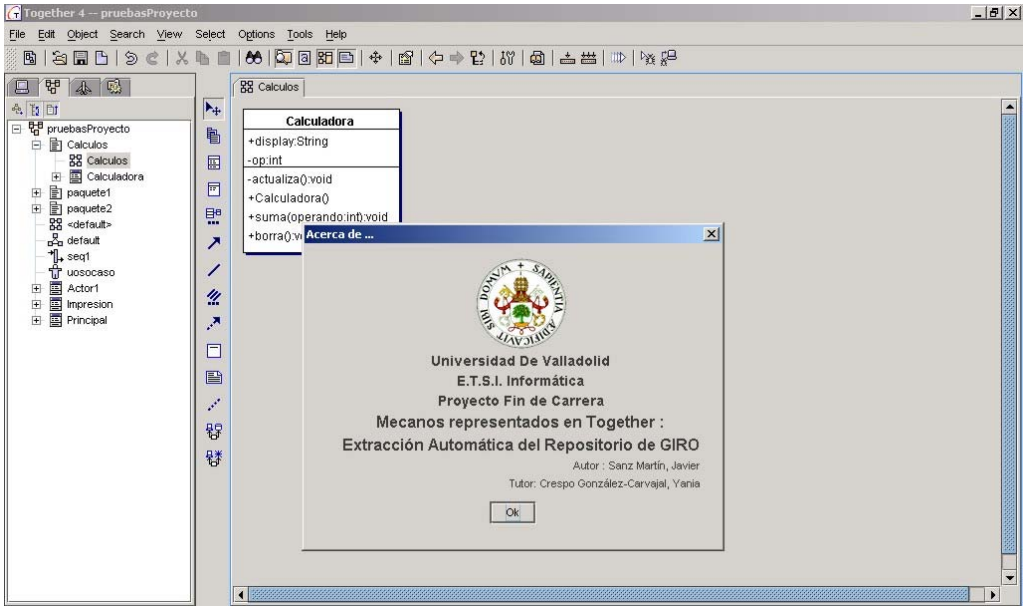

Figura 8.13. Acerca de...

8.2. Manual de Instalación

En este documento se detallan las acciones a realizar para que el usuario obtenga una versión operativa del sistema en su hardware.

8.2.1. Compilación

Para la compilación del sistema desarrollado, se necesita como prerrequisito contar el la máquina del usuario una instalación operativa de la herramienta CASE, pues el sistema entero ha sido desarrollado (análisis, diseño e implementación) con el entorno integrado de desarrollo proporcionado por Together. Es decir, el propio sistema en su fase de desarrollo fue un proyecto Together.

La compilación el Together se realiza a través del menú "*Tools*" en la opción "*Make Project*". Tras una orden de compilación, Together compilará las clases que componen el proyecto activo en Together informando al usuario de un eventual error en la compilación. El destino de las clases Java compiladas (archivos **.class*) puede personalizarse en el proyecto Together.

Previo paso a la compilación es necesario señalar que para el desarrollo del sistema se utilizaron una serie de bibliotecas externas distribuidas en archivos JAR. Son las que se detallan a continuación:

- *xml4j.jar* : Es una librería para el tratamiento (parseado) de ficheros XML. En concreto contiene la aplicación TXDOMParser, que es un parser XML que responde al modelo DOM. Con esta librería se realizará el procesamiento de los assets diagramas que contienen documentos XMI (XML). Together 4.2 proporciona en su distribución una copia de esta libería.
- *servlet.jar* : Esta librería permite compilar servlets, dando soporte para la aplicación servidor de este sistema.
- *jce.jar* : Es la extensión para Java 2 que da soporte a criptografía. Proporciona las interfaces de acceso a los métodos de encriptado.
- *bouncycastle_provider.jar* : Es un proveedor de seguridad (*Security Provider*) para la extensión JCE de Java. Esta librería contiene las implementaciones de los algoritmos de encriptado que se accesibles a través de la interfaz JCE.
- *interchange.jar* : Es el módulo de Together para intercambio de datos y entre sus funcionalidades está la importación de documentos XMI (viene con la distribución de Together en:

```
%DIRECTORIO_TOGETHER%/modules/com/togethersoft/modules/interchange
```

Este sistema también necesita de las bibliotecas de acceso a JDBC (*classes111.zip*) y al API de Together (*openapi.jar*), pero estas librerías se proporcionan con todas las distribuciones de Together.

A pesar de que el sistema hace uso de un analizador de código del lenguaje Java producido por la herramienta JavaCC, éste no se distribuye como librería, sino que JavaCC lo proporciona como fuentes. Esta librería de la que se disponen los fuentes se compilará y almacenará junto a este sistema en particular junto al cliente (que es quién lo utilizará) para evitar los problemas de pérdida o borrado de los componentes del sistema y facilitar su distribución.

La ubicación de estas librerías puede personalizarse, desde la opción de propiedades de un Proyecto en Together. Se recomienda encarecidamente al usuario que adapte las rutas de acceso a las librerías antes citadas a la ubicación que possen su distribución de Together particular antes de realizar una compilación.

Los fuentes de este sistema se distribuyen en dos grandes paquetes y una librería de análisis de código Java a compilar. Su estructura en paquetes es:

- *com.togethersoft.modules.ImportarMecanoGIRO* : Contiene al cliente y al paquete *Comunicaciones* que representa los datos de intercambio en comunicaciones entre el cliente y servidor.
- *PackageJavaParser* : Analizador de código obtenido como producto de JavaCC y una sintaxis reducida de Java.
- *ServidorImportar* : Contiene el Servidor del sistema que realizará los accesos al repositorio.

Una vez tenidas en cuenta estas directrices, para compilar el sistema, sólo hay que abrir desde Together el proyecto que contiene su desarrollo (proporcionado con el CD que acompaña e esta memoria) y seleccionar la opción de compilar desde los menús de Together antes explicados.

El proyecto Together que representa al sistema, almacenará las clases Java compiladas en el directorio:

```
%DIRECTORIO_PROYECTO%/classes
```

El dicho directorio se encuentran los archivos **.class* organizados en directorios respondiendo a la estructura de paquetes que los fuentes mantenían.

8.2.2. Instalación del sistema.

Una vez obtenidas las clases compiladas del sistema de detallará como debe realizarse su instalación en cada herramienta para su utilización.

8.2.2.1. Instalación del Cliente.

El cliente necesita para su funcionamiento una versión operativa de Together (4.2, para la que fue desarrollado o superior).

El cliente constituye un módulo Together y debe instalarse como tal. Teniendo en cuenta que los módulos de Together son distribuidos e instalados a través de archivos JAR, se deberá empaquetar el cliente en esta estructura. Los módulos JAR de Together deben contener un archivo *manifest* con información del la carga del módulo. En el apartado de implementación se optó por un archivo *manifest.mf* cuya descripción vuelve a presentarse:

```
Manifest-Version: 1.0
Name: Importar Mecano
Time: Startup
```

```
Main-Class:
com.togethersoft.modules.ImportarMecanoGIRO.ModuloImportarMecano
Created-By: Sanz Martin, Javier

Name: Importar Mecano
Time: User
Main-Class:
com.togethersoft.modules.ImportarMecanoGIRO.ModuloImportarMecano
Folder: "Mecano/ImportarMecano"
Created-By: Sanz Martin, Javier
HelpFile:"$TGH$/modules/com/togethersoft/modules/ImportarMecanoGIRO/he
lp/index.html"
```

Para almacenar los archivos class en un archivo JAR, Java dispone de una herramienta de uso en línea de comandos que permite generar un archivo JAR a partir de las clases compiladas y de el archivo *manifest*. Su sintaxis es:

```
jar cvfm <archivo.jar> <manifest> -C <ruta origen> <ruta almacenamiento en el
JAR>
```

<*archivo.jar*> el en nombre del archivo JAR a crear.
<*manifest*> es el nombre del archivo manifest empleado.
<*ruta origen*> es la ruta a partir de la cual serán los archivos class almacenados.
<*ruta almacenamiento en JAR*> es la ruta que se añadirá como prefijo a la ruta de cada archivo **.class* que sea almacenado.

Para fabricar el JAR para el cliente, el usuario deberá situarse en el directorio `%PROYECTO_TOGETHER%/classes`, donde se encontraban los ficheros *class*, y suponiendo la presencia del archivo *manifest.mf* en el directorio del proyecto de este sistema(`%PROYECTO_TOGETHER%`), y que se desea generar el archivo JAR *ImportarMecanoGIRO.jar* en ese mismo directorio raíz del proyecto la línea de comandos sería:

`jar cvfm ..\ImportarMecanoGIRO.jar ..\manifest.mf -C . .` (para plataformas Windows)
ó
`jar cvfm ../ImportarMecanoGIRO.jar ../manifest.mf -C . .` (para plataformas Unix o derivados)

La herramienta JAR viene distribuida de forma conjunta con cualquier versión del *Java Development Kit* (JDK). La distribución Together 4.2 para plataformas Windows viene junto al JDK 1.3.1. que se situará en:

```
%DIRECTORIO_TOGETHER%/jdk
```

Puede retirarse el directorio de las clases compiladas del servidor `%PROYECTO_TOGETHER%/classes/ServidorImportar` para la elaboración del JAR, puesto que el cliente no lo necesitará.

Una vez obtenido el JAR del Cliente, éste debe instalarse como módulo de Together. Si se tiene el cliente en el JAR *ImportarMecanoGIRO.jar*, para instalarlo cómo un módulo de Together deberá copiarse este archivo JAR al directorio:
`%DIRECTORIO_TOGETHER%/modules/com/togethersoft/modules/ImportarMecanoGIRO`,

así como la documentación de ayuda disponible para este módulo y que puede obtenerse del CD que acompaña a esta memoria. Dicha documentación esta en el directorio del CD `Proyecto/Instalación/Cliente` que además contiene una distribución ya compilada del Cliente *ImportarMecanoGIRO.jar* para una rápida instalación. Se debe copiar el directorio `help` de la anterior ruta , junto al cliente:

```
%DIRECTORIO_TOGETHER%/modules/com/togethersoft/modules/ImportarMecanoGIRO
```

Ahora de deben instalar en el sistema las librerías que necesita el cliente para su funcionamiento. Descritas en el apartado anterior de Compilación, sólo se describirá la ruta de su ubicación:

- *xml4j.jar* : Debe copiarse al directorio `lib` de Together, es decir: `%DIRECTORIO_TOGETHER%/lib`
- *jce.jar* : Debe situarse en el directorio de extensiones del directorio de librerías de la máquina virtual Java (JVM) instalada en la máquina del usuario. Este directorio se encuentra en : `%DIRECTORIO_JDK%/jre/lib/ext` y como anteriormente se mencionó la distribución para plataformas Windows de Together 4.2 incluye el JDK 1.3.1 en el directorio `%DIRECTORIO_TOGETHER%/jdk`
- *bouncycastle_provider.jar* : El proveedor de JCE debe situarse en el mismo directorio: directorio de extensiones del directorio de librerías de la máquina virtual Java (JVM), es decir, `%DIRECTORIO_JDK%/jre/lib/ext`. El proveedor de Seguridad contenido en esta librería no es necesario que se instale explícitamente en los archivos de configuración de Java, labor tediosa y puede que incompatible con la política de seguridad en la máquina en la que se instalará el cliente. Para evitar esto, la implementación del cliente llama explícitamente al proveedor de esta librería con lo que se evitan esa serie de problemas.

Todas estas librerías se encuentran disponibles en el CD que acompaña a esta memoria en el directorio `Proyecto/Instalación/Librerias` para su utilización. Añadir solamente que la librería *interchange.jar* por ser un módulo de Together ya instalado no es necesario instalarla.

Por último y para finalizar la instalación del cliente se ha de señalar que para que el módulo Together procese correctamente los documentos XML que tendrá que manejar, las DTDs que los definen (y que serán consultadas por la aplicación TXDOMParser) deben de estar disponibles para Together. Las DTDs necesarias se distribuyen en el CD que acompaña a esta memoria en `Proyecto/Instalación/DTDs`. Este directorio contiene dos DTDs *uml.dtd* y *mecano.dtd*, para procesar los documentos XMI y representación lógica de elementos reutilizables respectivamente. Para su instalación en Together deberán ser copiadas al directorio de ejecución de Together (desde donde arranca esta aplicación) que es:

```
%DIRECTORIO_TOGETHER%/bin
```

8.2.2.2. Instalación del Servidor

Para la correcta instalación y funcionamiento del servidor se ha de disponer en la arquitectura sobre la que se ejecute del motor de servlets Jakarta-Tomcat en estado operativo de funcionamiento.

Los servlets que ejecuta Tomcat deben situarse en un determinado directorio y con una cierta estructura de subdirectorios que se detalla a continuación.

Cada servlet de Tomcat viene definido por un directorio situado en el directorio:

```
%DIRECTORIO_TOMCAT%/webapps/<nombre servlet>
```

Cada servlet definido por el anterior directorio, tendrá la siguiente estructura de subdirectorios:

- *Log* : Directorio donde se almacenarán en archivos las salidas estándar y de error.
- *Meta-inf*: Información acerca de autores, versión, etc, presente en un archivo *manifest* (cuya estructura ya fue explicada)
- *Web-inf*: éste será el directorio que contenga las clases compiladas del servidor.

Las clases de servidor no pueden ser copiadas simplemente en ese diectorio, pues este directorio también tiene una estructura definida:

- *web.xml* : Es un archivo de configuración para Tomcat del servlet en cuestión. Indica la clase raíz, redirección de salidas y por supuesto el URL que atenderá este servlet.
- *classes* : Es el directorio donde se almacenarán las clases compiladas del servlet. Estas deberán conservar su estructura propia de paquetes a fin de que el servlet sea operativo.

Para el caso concreto del servlet desarrollado, la descripción del archivo *web.xml* fue detallada con anterioridad en el documento de implementación.

Puesto que esta estructura de directorios y subdirectorios puede parecer extraña y complicada al usuario, en el CD que acompaña a esta memoria se encuentra en el directorio `Proyecto/Intalacion/Servidor` una distribución compilada del servidor con toda la estructura de directorios antes descrita. Bastará con copiar al directorio: `%DIRECTORIO_TOMCAT%/webapps` el contenido del directorio del CD descrito anteriormente `Proyecto/Intalacion/Servidor`.

Una vez efectuada esta labor, para que el servlet sea operativo se han de instalar en Tomcat todas las librerías que el servidor utilice. Estas fueron descritas con anterioridad en el apartado de Compilación y son :

- *servlet.jar* : Debe copiarse al directorio `lib` de Tomcat, es decir: `%DIRECTORIO_TOMCAT%/lib`
- *jce.jar* : Debe situarse en el directorio de extensiones del directorio de librerías de la máquina virtual Java (JVM) instalada en la máquina del servidor. Este directorio se encuentra en : `%DIRECTORIO_JDK%/jre/lib/ext`

- *bouncycastle_provider.jar* : El proveedor de JCE debe situarse en el mismo directorio: directorio de extensiones del directorio de librerías de la máquina virtual Java (JVM), es decir, `%DIRECTORIO_JDK%/jre/lib/ext`. El proveedor de Seguridad contenido en esta librería no es necesario que se instale explícitamente en los archivos de configuración de Java, labor tediosa y puede que incompatible con la política de seguridad en la máquina en la que se instalará el servidor. Para evitar esto, la implementación del servidor llama explícitamente al proveedor de esta librería con lo que se evitan esa serie de problemas.
- *ImportarMecanoGIRO.jar* : El servidor necesita conocer los datos de intercambio que él y el cliente emplearán durante las comunicaciones. Descrito desde el Documento de Diseño, estos datos de intercambio eran el paquete Comunicaciones que a nivel de implementación se situó como un paquete dentro del cliente. En la elaboración del módulo Togeter en JAR lo que se hizo es dotarle de una estructura de librería para su utilización. Dado que esta librería cliente contiene el paquete Comunicaciones necesario para el intercambio de datos con el cliente, se usará como una librería. La librería cliente debe copiarse al directorio `lib` de Tomcat, es decir: `%DIRECTORIO_TOMCAT%/lib`

Todas estas librerías se encuentran disponibles en el CD que acompaña a esta memoria en el directorio `Proyecto/Instalación/Librerias` para su utilización.

8.2.2.3. Instalación de JCE y el Proveedor de Seguridad.

Se ha hecho referencia en la instalación de cliente y servidor de la necesidad de instalar dentro de los directorios de librerías del Proveedor de Seguridad y del la extensión JCE para Java.

Dicha necesidad nace de que estas librerías son extensiones de Java, que la máquina virtual java ha de conocer e incluso configurar.

La configuración del Proveedor de Seguridad para la extensión se realiza en el archivo java.security situado en el directorio %`DIRECTORIO_JDK%/jre/lib/security`.

La estructura de este archivo se hace en base a `%atributo%=%valor%` como en otros archivos de configuración vistos en esta memoria. En concreto, para instalar un Proveedor de Seguridad para la máquina virtual Java (para todas sus ejecuciones, no para una aplicación en concreto) se añade una línea al archivo *java.security* en la primera sección del mismo que sea del tipo:

```
security.provider.<prioridad de uso>=<nombre completo (cualificado) de la clase raíz del proveedor>
```

que para este caso en particular en que se eligió el proveedor BouncyCastle de libre distribución se tendría que para usuarlo como el primer proveedor de seguridad habría que colocar en la línea anterior:

```
security.provider.1=org.bouncycastle.jce.provider.BouncyCastleProvider
```

Este acceso y modificación de la máquina virtual Java puede no estar permitido por la política de seguridad del la instalación de la máquina virtual Java, o puede no ser deseable para el sistema. Por ello, tanto el cliente como el servidor no necesitan de la declaración de ningún proveedor de seguridad en la máquina virtual. Cuando necesitan

de él, lo llaman explícitamente (a la clase raíz del mismo), con lo que se evitan los problemas citados anteriormente. Sólo es necesario situar al proveedor y a la extensión JCE accesibles a la máquina virtual, que se consigue al copiar las librerías que los contienen al directorio de extensiones del directorio de librerías de la máquina virtual Java, `%DIRECTORIO_JDK%/jre/lib/ext`.

El proveedor de seguridad y la extensión JCE dependen de la versión de máquina virtual de que se disponga en las plataformas del cliente y servidor.

Por ello, en el directorio de librerías del CD que acompaña a esta memoria se han adjuntado distribuciones de la extensión JCE, del Proveedor de seguridad BouncyCastle y de la licencia de uso de los mismos (requerido por la propia licencia de uso de este software) para las versiones de la máquina virtual Java:

- `JDK1.2.2 : Proyecto/Instalación/Librerias/JCE/jdk1.2.2`
- `JDK1.3.1 : Proyecto/Instalación/Librerias/JCE/jdk1.3.1`
- `JDK1.4.1 : Proyecto/Instalación/Librerias/JCE/jdk1.4.1`

9. DOCUMENTO DE PRUEBAS

9.1. Ámbito

En este documento se recogen las pruebas a las que fue sometido el sistema durante su construcción para probar su comportamiento al responder a las funcionalidades para la que fue diseñado.

9.2. Pruebas de Unidad

Se presentan ahora las pruebas realizadas para comprobar si el sistema respondía a la funcionalidad para la que fue creado. Las pruebas realizadas para esto fueron basadas en caja negra.

A continuación se lista una breve muestra de las pruebas efectuadas sobre el sistema, mostrando una única prueba por funcionalidad probada.

9.2.1. Pruebas genéricas o comunes al sistema

Conexión con el servidor	Número 1
Descripción	Al arrancar el módulo en el cliente comprueba la presencia de servidor en su localización estándar.
Entrada	Tanto servidor como cliente están activos.
Salida	La conexión se produce y se informa al usuario de la presencia del servidor en una dirección determinada.
Resultado	Correcto.

Conexión con el servidor	Número 2
Descripción	Al arrancar el módulo en el cliente comprueba la presencia de servidor en su localización estándar.
Entrada	El servidor no se encuentra activo
Salida	La conexión es rechazada y se informa al usuario de que en la dirección predeterminada no hay ningún servidor atendiendo peticiones.
Resultado	Correcto.

Descarga de assets	Número 3
Descripción	Comprobar que la representación física de un asset en el

	repositorio y la obtenida tras la descarga son la misma.
Entrada	Representación física de un asset en el repositorio.
Salida	Contenido de la representación física descargada.
Resultado	Correcto.

Unificación de assets diagramas	Número 4
Descripción	Comprobar que el la representación física de los assets diagramas y el documento XMI unión de yodos ellos contiene la misma información.
Entrada	Representación física de los assets diagramas.
Salida	Contenido del documento XMI unión de todos ellos
Resultado	Correcto.
Comentarios	Sólo ha sido comprobado para la herramienta Together 4.2

Autentificación de clave pública	Número 5
Descripción	Comprobar si la clave pública que reciben los clientes del servidor está autentificada.
Entrada	La clave pública enviada por el servidor y la firma de la misma.
Salida	Mensaje informando al usuario de que la seguridad está activada
Resultado	Correcto.

Encriptación	Número 6
Descripción	Comprobar que la encriptación de la identificación de usuario funciona
Entrada	Identificación de usuario encriptada mandada al servidor
Salida	Mensaje de Error o confirmación en función de que la identificación fuera válida o no.
Resultado	Correcto.

Proyecto sobre el que importar	Número 7
Descripción	Intentar importar un elemento reutilizable del repositorio sobre ningún proyecto abierto en Together
Entrada	Selección de las opciones que importan elementos reutilizables (Assets o Mecanos)
Salida	Se informa al usuario de que no es posible importar elementos reutilizables si no hay un proyecto Together abierto sobre el que importar
Resultado	Correcto.

Para el recorrido de las pruebas que afectan a la funcionalidad del sistema se hará en base a los casos de uso que cubrían dichas funcionalidades.

9.2.2. CS-1.01 Cargar módulo

Carga del módulo de Together	Número 8
Descripción	Cargar la aplicación cliente como un módulo de Together
Entrada	Iniciar la herramienta CASE
Salida	Se informa al usuario en la ventana de mensajes de que el módulo esta operativo al realizar una prueba de comunicaciones con el servidor.
Resultado	Correcto.

Carga del módulo de Together	Número 9
Descripción	Cargar la aplicación cliente como un módulo de Together
Entrada	Arrancar el cliente, una vez Together ya arrancado como un módulo de usuario (Opción *run* de la pestaña de módulos Together)
Salida	Se informa al usuario en la ventana de mensajes de que el módulo esta operativo al realizar una prueba de comunicaciones con el servidor.
Resultado	Correcto.

9.2.3. CS-1.02 Cambiar localización del Servidor

Carga del módulo de Together	Número 10
Descripción	Comprobar la personalización de la dirección del servidor al que el cliente se conecta
Entrada	Nueva dirección URL del servidor al que el cliente se conectará.
Salida	La dirección URL se cambió, y se realiza una prueba de comunicación con el servidor que atiende esa dirección
Resultado	Correcto.

9.2.4. CS-1.03. Cerrar sesión

Cierre de sesión	Número 11
Descripción	Cerrar la sesión de un usuario de la aplicación
Entrada	Selección y confirmación del cierre de sesión
Salida	Sesión cerrada
Resultado	Correcto.

Cierre de sesión	Número 12
Descripción	Cerrar la sesión de un usuario de la aplicación
Entrada	Selección y cancelación del cierre de sesión
Salida	Sesión no cerrada
Resultado	Correcto.

Cierre de sesión	Número 13
Descripción	Cerrar la sesión de un usuario de la aplicación
Entrada	Cierre de la herramienta Together y su posterior arranque de nuevo
Salida	Sesión cerrada
Resultado	Correcto.

9.2.5. CS-1.04 Validar Usuario

Identificar usuario	Número 14
Descripción	Introducir en la ventana de identificación el *login* y *password.*
Entrada	Login: m Password : *
Salida	Se informa al usuario de que ha sido identificado correctamente y que continúa la ejecución de la aplicación.
Resultado	Correcto.
Comentarios	El usuario "m" es la identificación estándar en el repositorio de la que se disponía durante la fase de desarrollo.

Identificar usuario	Número 15
Descripción	Introducir en la ventana de identificación el *login* y *password.*
Entrada	Login: ABCDE Password : FGHIJ
Salida	Se informa al usuario de que no puede acceder al sistema, su identificación es errónea, se cancela la operación en ejecución de la aplicación.
Resultado	Correcto.

9.2.6. CS-1.04 Buscar Proyecto

Búsqueda de Proyecto	Número 16
Descripción	Buscar un elemento reutilizable en el repositorio.
Entrada	No se introduce ningún datos en el patrón de búsqueda y se confirma que se desean buscar todos los elementos reutilizables de un tipo (asset o mecano) en el repositorio.
Salida	El cliente muestra todos los elementos reutilizables del repositorio en un listado
Resultado	Correcto.

Búsqueda de Proyecto	Número 17
Descripción	Buscar un elemento reutilizable en el repositorio.
Entrada	Se introduce un valor en concreto para algún campo de búsqueda.
Salida	Todos los elementos reutilizables listados contienen en esa propiedad en valor indicado.
Resultado	Correcto.

Búsqueda de Proyecto	Número 18
Descripción	Buscar un elemento reutilizable en el repositorio.
Entrada	Se introduce en el campo de búsqueda un valor que no contienen ninguno de los elementos del repositorio.
Salida	Se informa al usuario de que no se encontró ningún elemento reutilizable que cumpla esas características indicadas
Resultado	Correcto.

9.2.7. CS-1.06 Seleccionar Proyecto a Importar

Selección de Proyecto	Número 19
Descripción	Seleccionar un elemento reutilizable en el repositorio para su descarga.
Entrada	El usuario selecciona un elemento reutilizable de la lista.
Salida	El elemento se resalta sobre el resto
Resultado	Correcto.

Selección de Proyecto	Número 20
Descripción	Seleccionar un elemento reutilizable en el repositorio para su descarga.
Entrada	El usuario selecciona ver las propiedades del elemento reutilizable seleccionado en la lista .
Salida	Se presenta al usuario la información de elemento.
Resultado	Correcto.

Selección de Proyecto	Número 21
Descripción	Seleccionar un elemento reutilizable en el repositorio para su descarga.
Entrada	El usuario selecciona volver al listado de elementos reutilizables desde las presentaciones de propiedades de uno de ellos.
Salida	Se vuelve a mostrar al usuario el listado de elementos reutilizables
Resultado	Correcto.

Selección de Proyecto	Número 22
Descripción	Seleccionar un elemento reutilizable en el repositorio para su descarga.
Entrada	El usuario selecciona un elemento reutilizable para su descarga inmediata.
Salida	El cliente lanza una petición al servidor (visible como mensaje desde la ventana de mensajes de Together) de descarga del elemento reutilizable indicado.
Resultado	Correcto.

9.2.8. CS 1.08 – Importar Asset

Importar Asset	Número 23
Descripción	Importar un asset en el proyecto Together abierto.
Entrada	El usuario selecciona un asset para su importación (que previamente había sido seleccionado por un patrón de búsqueda indicado por el propio usuario)
Salida	El asset (con todos los assets necesarios para su reutilización) es importado en el proyecto abierto de Together
Resultado	Correcto.

Importar Asset	Número 24
Descripción	Importar un asset en el proyecto Together abierto.
Entrada	El usuario selecciona un asset para su importación (que previamente había sido seleccionado por un patrón de búsqueda indicado por el propio usuario)
Salida	La base de datos es inconsistente al no existir dicho asset físicamente, el sistema informa del error y cancela la operación
Resultado	Correcto.
Comentarios	El sistema presenta robustez ante los fallos

9.2.9. CS 1.09 – Importar Mecano

Importar Mecano	Número 25
Descripción	Importar un mecano en el proyecto Together abierto.
Entrada	El usuario selecciona un mecano para su importación (que previamente había sido seleccionado por un patrón de búsqueda indicado por el propio usuario)
Salida	El mecano (con todos los assets necesarios para su reutilización) es importado en el proyecto abierto de Together
Resultado	Correcto.

Importar Mecano	Número 26
Descripción	Importar un mecano en el proyecto Together abierto.
Entrada	El usuario selecciona un mecano para su importación (que previamente había sido seleccionado por un patrón de búsqueda indicado por el propio usuario)
Salida	La base de datos es inconsistente al no existir físicamente ninguno de los assets que forman parte del mecano, el sistema informa del error y cancela la operación
Resultado	Correcto.
Comentarios	El sistema presenta robustez ante los fallos

9.3. Otras pruebas

9.3.1. Pruebas de comunicaciones

Estas pruebas sirven para comprobar si las interfaces de los componentes del sistema tienen un funcionamiento correcto.

Tanto la interfaz entre el servidor y el repositorio como las comunicaciones entre servidor y cliente a través del componente del sistema Comunicaciones quedan probadas por las pruebas anteriores referidas a la funcionalidad del sistema.

9.3.2. Pruebas de facilidad de uso.

La interfaz del sistema con el usuario es tan intuitiva y agradable (iconos, botones, etc) que estas pruebas quedan cubiertas.

En las pruebas anteriores efectuadas sobre el interfaz de la aplicación Together, así como en el manual de usuario puede observarse esta facilidad de manejo de la aplicación

9.3.3. Prueba de documentación y ayuda

Tanto la ayuda proporcionada por la documentación de usuario ya fuera para uso o instalación, así como la distribución del sistema en fuentes y en librerías compiladas hacen que cualquier usuario según su nivel de experiencia pueda usar e instalar este sistema software.

Especialmente la documentación de uso ha sido revisada varias veces y se han incluido ejemplos del resultado de aparecerá en el interfaz gráfico de usuario ante la realización de una operación.

Por todas estas características las pruebas de documentación y ayuda quedan probadas.

10. CONLUSIONES Y LÍNEAS DE TRABAJO FUTURO

Con este sistema software desarrollado se dota la al herramienta Together (junto con otras aplicaciones desarrolladas con anterioridad) de soporte para desarrollo con y para reutilización. Ahora Together se convierte en una herramienta muy potente para el desarrollo de sistemas software bajo Java, al permitir el uso de componentes ya fabricados. Esto supondrá que los productos software desarrollados con Together y reutilización serán mas fiables y robustos, a la vez que más baratos, al haberse construido en menos tiempo.

Para futuros trabajos sería interesante que tanto la aplicación desarrollada en este sistema (desarrollo con reutilización) como las aplicaciones de anteriores trabajos (desarrollo para reutilización) se fusionaran en una sola, para facilitar su distribución e integración de su funcionamiento. En cualquier caso, esa tarea cae lejos de los objetivos de la aplicación expuesta, pero se deja en el aire esa opción: integrar las aplicaciones orientadas a reutilización bajo Together en una sola. Es mas, se podría pensar en un servidor centralizado que accediera al repositorio y que atendiera las peticiones de múltiples clientes embebidos en distintas herramientas CASE, con lo que la extensión de la reutilización llegaría a cotas más que deseables.

Por último se recomienda profundizar en el tema de la seguridad en las comunicaciones entre los clientes y el servidor que accede al repositorio. Con este proyecto se pretendía dibujar un esbozo de lo que podría ser una comunicación que garantice privacidad e integridad de las transmisiones a todos los niveles. Como primer paso desde este proyecto se invita a profundizar el las ventajas que el encriptado basado el clave pública para Java proporciona para estos fines, así como las múltiples funcionalidades que proporciona el proveedor de seguridad para Java elegido, para su uso (dado su nulo coste y opción de código fuente abierto) en futuros proyectos o aplicaciones que requieran comunicaciones seguras en el ámbito del desarrollo software.

APÉNDICES

En éste último apartado se describirán los temas que sin relación directa con el objetivo de este proyecto, pero que han condicionado su diseño o implementación.

Aparte de las consabidas herramientas que implica el desarrollo de este sistema (Together, JDBC, Java, etc.), este documento pretende presentar los nuevos conceptos empleados en este sistema (seguridad y criptografía) sin olvidar el núcleo base de esta aplicación y del resto de aplicaciones desarrolladas desde GIRO para favorecer la reutilización:

Criptografía

La criptografía

La criptografía o cifrado de la información tiene una historia muy larga, desde sus iniciales y rudimentarios usos en el Egipto de hace 4000 años hasta su moderna acepción el siglo XX (donde jugo un papel crucial en ambas guerras mundiales) y XXI.

Su verdadero auge surge en los años 60 del pasado siglo, cuando ante la proliferación de los ordenadores y las comunicaciones a través de ellos, el sector privado de la industria comenzó a demandar soportes para almacenar sus datos en ordenadores y protegerlos de extraños mediante sistemas de seguridad.

La protección de los datos anteriormente descritos cubriría cuestiones ente otras del tipo:

- **Privacidad** : La información debe mantenerse secreta para todos aquellas personas o entidades no autorizadas a conocerla
- **Integridad de los datos** : Seguridad de que la información no fue alterada por medios no autorizados o desconocidos.
- **Identificación** : Correspondencia entre entidad que usa un sistema y su identidad.
- **Autentificación de la información** : Confirmación de que la fuente de la información es quien dice ser.
- **Firmado** : Los medios para asociar una información a la entidad que la distribuye.
- **Autorización** : Permiso oficial expreso de una entidad a realizar alguna acción.

- **Validación** : Los medios proporcionados por un sistema para que una entidad pueda manipular información o recursos
- **Control de acceso** : Restricción del acceso a unos recursos sólo usables por entidades privilegiadas.
- **Certificación** : Información obtenida de una entidad de confianza.
- **Anonimato** : La ocultación de la identidad de una entidad mientras esta realice alguna tarea concreta.

Una vez fijados los objetivos que la criptografía debe cumplir, se la puede definir como el estudio de las técnicas matemáticas relacionadas con aspectos de la seguridad de la información como confidencialidad, integridad de datos, identificación de entidades y autentificación del origen de datos.

Las gran variedad de herramientas que proporcionará la criptografía deben ponderarse en función de la tarea que se desee llevar a cabo y unos buenos criterios para realizar este peso de las opciones serían:

- **Nivel de seguridad**: Se define en términos del número de operaciones a realizar para poder romper un sistema de seguridad. Normalmente está definido por una cota superior del trabajo computacional necesario para romper el sistema de seguridad.
- **Funcionalidad**: Las primitivas o acciones simples que deben realizarse para cubrir el objetivo de seguridad.
- **Modos de funcionamiento**: El modo o la forma de aplicar las primitivas supondrá obtener un sistema de seguridad con unas características distintas.
- **Rendimiento** : Hace referencia a la eficiencia de las primitivas empleadas.
- **Facilidad de Implementación** : hace referencia a la dificultad para llevar una primitiva a una implementación en una computadora.

Actualmente es uno de los campos de estudio intenso en la computación, dada su gran repercusión en los sistemas de comunicación y comercio a través de medios informáticos.

Tipos de sistemas criptográficos

En las primitivas de criptografía al información es tratada para su seguridad mediante "c*laves*". Si a una información se le aplica una primitiva criptográfica con una determinada clave, esta se convertirá en una información encriptada, carente de todo sentido para una entidad ajena al sistema de información que se pretende asegurar, y que sólo podrá ser recobrada mediante la aplicación de otra primitiva de encriptado asociada a otra clave.

Atendiendo a la forma en que la información es tratada para proporcionar seguridad en su transmisión, se podrían dividir actualmente los sistemas de criptografía en dos grandes familias:

Criptografía basada en clave simétrica : Las claves que proporcionan seguridad al sistema de información mediante su uso en primitivas de encriptación (primitivas para ocultar la información) y en primitivas de desencriptación (primitivas para recobrar una información previamente encriptada), pueden ser fácilmente determinables (computacionalmente hablando) la una en función de la otra. Para estos sistemas normalmente la clave usada en las primitivas de encriptación coincide con la clave usada en las primitivas de desencriptación. Suelen recibir otros nombres como sistemas de "*clave simple*" o "*clave privada*".

Criptografía basada en clave pública : Las claves que proporcionan seguridad al sistema de información mediante su uso en primitivas de encriptación (primitivas para ocultar la información) y en primitivas de desencriptación (primitivas para recobrar una información previamente encriptada), no pueden ser determinadas computacionalmente hablando la una a partir de la otra. Estos sistemas pueden parecer los deseables de usar, pero esta característica que los define tiene importantes consecuencias a la hora de su utilización como se explica a continuación. Los sistemas criptográficos basados en clave pública se componen de dos claves empleadas en las primitivas para asegurar la información en la trasmisión entre dos entidades: la clave pública (que le da nombre) y la clave privada. Las funciones de ambas son intercambiables (lo encriptado con una se recupera con la otra y viceversa).

Supongamos un sistema de transmisión de información a través de un canal de comunicación que no proporciona seguridad a ésta comunicación entre dos entidades A y B. Para una transmisión de A a B por este canal inseguro, B utilizará un sistema de encriptado basado en clave pública. La clave de encriptado recibe el nombre de clave pública y es conocida por la entidad A por todas las entidades que pretendan comunicar información a B de forma segura. Mediante la aplicación de una primitiva de encriptado basada en clave pública con la clave pública del sistema criptográfico que A conoce, encripta el mensaje de información que pretende mandar a B. Ahora este mensaje puede mandarse por el canal de comunicación no seguro que comunica a A con B. Para que B pueda recobrar el mensaje original que A pretendía mandarle, del mensaje encriptado, B aplica una primitiva de desencriptado con la clave privada. Dicha primitiva al aplicarla con la clave privada (únicamente conocida por B) sobre el mensaje que envió A encriptado proporciona el mensaje original.

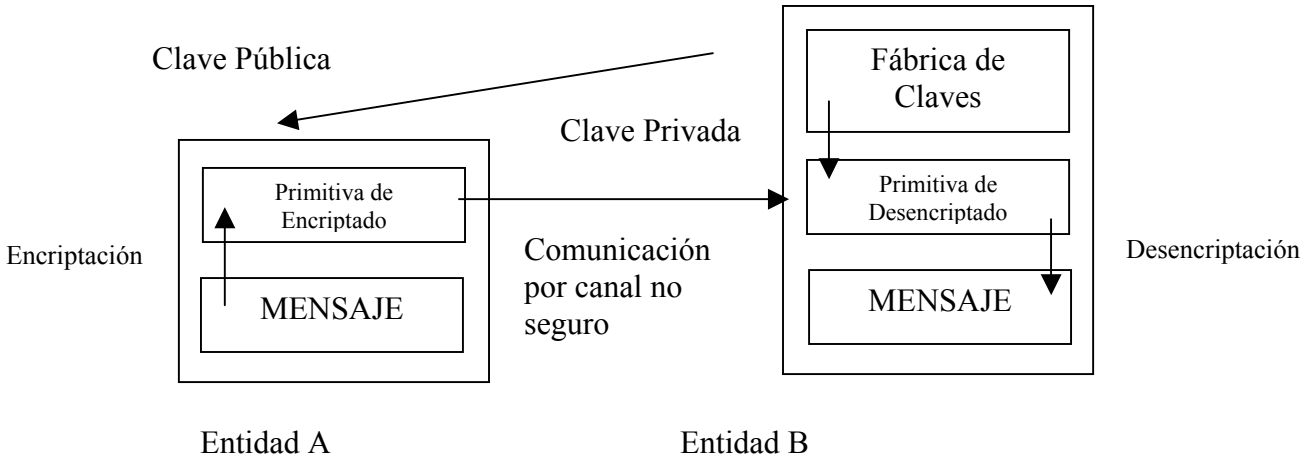

Figura A.1. Esquema de encriptación basado en clave pública.

Mediante el sistema antes expuesto se obtiene una comunicación segura de A a B. Mediante el empleo de otro por de claves y un sistema criptográfico basado en clave pública (que puede ser el mismo) se obtendría comunicación segura de B a A.

Este sistema de criptografía presenta un problema: cómo puede obtener A la clave pública del sistema criptográfico empleado por B. Si esta se envía por el canal inseguro de comunicación, alguna entidad externa hostil podía situarse entre ambas entidades en comunicación, suministrarle a A una clave pública propia y A creyendo que el remitente de dicha clave era B, enviar por el canal de comunicación encriptados con la calve falsa, con lo que la entidad hostil podría conocer el contenido del mensaje de A.

Para esto se emplea autentificación de las claves públicas (o firmas digitales) en estos sistemas de encriptado. Su sistema es muy parecido al encriptado con clave pública, pero los roles de emisor y receptor se invierten. Supongamos que los mensajes que enviará B a A siguen una estructura que ambos conocen *a priori.* Para que A obtenga un mensaje de B (como una clave pública por ejemplo) con la seguridad de que B es su remitente se realizan las siguientes acciones. B encripta un mensaje (cuya estructura es conocida *a priori* por A y B) con su primitiva de desencriptado y su clave privada del sistema de comunicaciones expuesto con anterioridad. Dicho mensaje es mandado a través del canal de comunicaciones no fiable, con la seguridad de que nadie conocerá su contenido (pues está encriptado). Cuando A recibe dicho mensaje encriptado le aplica su primitiva de encriptado (del sistema de comunicaciones anterior) con la clave pública de B. Puesto que los roles de las claves pública y privada son intercambiables, lo encriptado con la clave privada se recupera con la clave pública, luego A obtiene el mensaje sin encriptación mandado por B. Si el mensaje obtenido se ajusta a la estructura de mensajes que B intercambia con A, A tiene la seguridad de que fue realmente fue B el remitente del mensaje. Esto se conoce con el nombre de *Firmas digitales con encriptación reversible de clave pública* y puede servir por ejemplo para el problema de la autentificación de las claves públicas enviadas por un canal de comunicaciones inseguro.

Ventajas e inconvenientes de ambos tipos de encriptado

Las ventajas e inconvenientes más importantes que estas dos familias de sistemas de encriptado son:

Ventajas de la clave simétrica:

- Los sistemas de encriptado de este tipo tienen una eficiencia mayor, proporcionado anchos de banda en sus implementaciones de entre cientos de MB/s (por hardware) y MB/s (por software).
- Las claves de estos sistemas, claves simétricas son relativamente cortas.
- Sistemas de encriptado simples basados en clave privada pueden componerse para formar sistemas de encriptado más complejos y robustos.
- Los sistemas de encriptado de clave simétrica tienen una larga historia y son bien conocidas sus carencias y virtudes frente a los relativamente nuevos (nacidos en la década de los 70s del pasado siglo) sistemas de encriptado basados en clave pública.

Inconvenientes de la clave simétrica:

- En una comunicación entre dos entidades, esta debe ser mantenida en secreto por los dos.
- Para redes de comunicaciones grandes el número de claves a mantener es muy elevado.
- En una comunicación entre dos entidades, la clave debe ser cambiada cada cierto tiempo para garantizar que se mantiene en secreto.
- Los métodos de firma digital basados en clave simétrica requieren claves muy grandes.

Ventajas de la clave pública:

- Sólo debe mantenerse la clave privada en secreto y por una única entidad.
- La administración de las claves sobre una gran red, sólo requiere de una entidad de confianza que mantenga las claves y que distribuya las calves públicas con autentificación de las mismas.
- El ciclo de vida de las claves públicas y privadas de estos sistemas es más largo que para las claves simétricas.
- Estos sistemas permiten implementar mecanismos de firma digital eficientes.

Inconvenientes de la clave pública:

- La eficiencia y el ancho de banda son cientos de veces menores que sus equivalentes en clave simétrica.

- El tamaño de las claves es normalmente mucho mayor que el de las claves simétricas (del orden de 1024 bits para clave publica y 64-128 bits para clave simétrica).
- El sistema de clave simétrica ha demostrado ser seguro, mientras que los sistemas de clave pública basan su seguridad en la presunta dificultad computacional de una serie de problemas teóricos matemáticos.
- Es relativamente nueva y desconocida (fue descubierta en la década de 1970)

Un caso particular: RSA

El algoritmo de encriptado basado en clave pública RSA fue en primer algoritmo de este tipo descubierto. Publicado por sus descubridores (R. Rivest, A. Shalmir y L. Adleman, de ahí viene su nombre-RSA) en 1977, proporciona privacidad y firmas digitales en las comunicaciones.

Sin embargo, la técnica básica de este algoritmo fue descubierta en 1973 or el británico Clifford Cocks, pero que fue mantenido en secreto hasta 1997.

Su seguridad está basada en la intratabilidad computacional de la factorización de enteros, la cual permitiría obtener para este algoritmo la clave privada de encriptado a través de la clave pública (conocida esta deficiencia como el *problema RSA*).

Su funcionamiento queda descrito:

Generación de claves:

1. Se generan dos números primos grandes p y q de tamaño aproximado cuyo producto $n=p \cdot q$ es del tamaño de clave requerida (normalmente 1024 bits).
2. Se computa el entero *n*, o módulo, $n=p \cdot q$ y $phi = (p-1) \cdot (q-1)$
3. Se escoge un entero *e*, conocido como exponente público (o de encriptación), tal que $1 < e < phi$ que cumpla que su máximo común divisor con *phi* sea 1 (son primos entre sí), $mcd(e, phi)=1$.
4. Se computa un entero *d*, conocido como exponente secreto (o de desencriptado), tal que $1 < d < phi$ y que cumpla que $e \cdot d \equiv 1 (mod\ phi)$.
5. La clave pública es (n, e) y la clave privada es (n, d)

Encriptado:

Para que una entidad A envíe un mensaje a otra entidad B:

1. A obtiene la clave pública de encriptado de B (n,e).
2. A representa el mensaje a enviar como un entero $m < n$ (puede dividir el mensaje en pequeños mensajes m' del tamaño requerido tal que $m' < n$).
3. Se computa el mensaje encriptado $c=m^e \bmod n$.
4. A envía el menaje encriptado c a B.

Desencriptado:

Una vez obtenido el mensaje encriptado c de A, B realiza las siguientes operaciones:

1. B recupera el mensaje mediante el cómputo de $m= c^d \bmod n$
2. B representa el mensaje desde el entero m de la misma manera que A representó el mensaje como un entero.

La Patente del RSA

El uso del algoritmo RSA estaba sometido a patente, patente que se conocía por el propio nombre del algoritmo. Fue editado el 20 de Septiembre de 1983, pero creada su patente el 17 de diciembre del 1977 (pat. #4405829) y asignada por sus creadores al Massachussets Institute of Technology. La patente cubría la encriptación basada en clave pública y el método de firma digital. La leyes de EE.UU. indican que una patente tiene una validez de 17 años desde la fecha de su edición o 20 años desde la fecha en que una solicitud de patente fue rellenada. Para solicitudes anteriores al 8 de junio de 1995 (y no caducadas a esa fecha) se aplicará el periodo más largo. Teniendo en cuenta estas consideraciones legales, se tiene que la patente del RSA caducó el 20 de septiembre del año 2000, y que ahora su utilización es libre.

La estructura del Repositorio de GIRO

El repositorio de GIRO se encuentra implementado sobre una base de datos Oracle8i. El acceso a la información que contiene (representación lógica de assets y mecanos, así como las relaciones que los enlazan) fueron de estudio prioritario para el desarrollo del sistema.

La estructura de tablas que definen al repositorio son:

Nombre de la tabla	Campos
Analisis	idAnalisis
Asset	idAsset idTipoAsset metodo idNivelAbstraccion
Cualificacion	idcualificacion url observaciones idasset
Disenno	iddisenno
Dominio	iddominio nombre descripcion
ElementoReutilizable	idelementoreutilizable nombre resumen fechacreacion fechaintroduccion fechamodificacion palabrasClave plataforma restricciones idnivelseguridad coste ididioma version idnivelevaluacion autores palabras
Implementacion	idimplementacion entorno contenido idsistemaoperativo idlenguaje plataforma compilador idred idsgbd operaciones observaciones
Lineaproducto	idlineaproducto

	nombre descripcion iddominio
Mecano	idmecano paradigma
Fachada	idfachada nombre descripcion idmecano
Organización	idorganizacion nombre razonsocial direccion ciudad codigopostal idpais email telefono fax comentarios
Origendestino	idorigendestino idtipoassetorigen idtipoassetdestino idtiporelacion
Relacion	idrelacion nombre idassorigen idassdestino idtiporelacion
Representacion	idrepresentacion Nombre url idformato tamanno idmedio comentarios
Usuario	idusuario nombre apellido1 apellido2 idtipodocumento documento direccion ciudad codigopostal idpais email telefono fax comentarios identificacion clave rol

	idorganizacion

Y las tablas básicas auxiliares de consulta fueron:

Nombre de la tabla	**Campos**
FormatoFichero	idformaton nombre
Idioma	ididioma nombre
Lenguaje	idlenguaje nombre
Medio	idmedio nombre
NivelAbstraccion	idnivelabstraccion nombre descripcion
NivelEvaluacion	idnivelevaluacion nombre
NivelSeguridad	idnivelseguridad nombre
Pais	idpais nombre
PalabrasClave	idpalabraclave palabra
Red	idred nombre
Sgbd	idsgbd nombre
SistemaOperativo	idsistemaoperativo nombre
Tipoasset	idtipoasset nombre paradigma
TipoDocumento	idtipodocumento nombre
TipoEstructural	idtipoestructural nombre confiorigen confidestino
TipoRelacion	idtiporelacion nombre esinternivel idtipoestructural

BIBLIOGRAFÍA

Artículos y trabajos

🕮 ANDÚJAR PUERTAS, JULIÁN. NISTAL CALVO, ALBERTO. *Mecanos representando proyectos Together: Inserción Automática en el repositorio de GIRO.* Proyecto Fin de Carrera. Universidad de Valladolid, Julio 2002.

🕮 DI MANAGEMENT SERVICES. *RSA Algorithm*.

http://www.di-mgt.com.au/rsa_alg.html

🕮 FLINN, PATRICK J. JORDAN, JAMES M. *Using the RSA algorithm for Encryption and Digital Signatures : Can You Encrypt, Decrypt, Sign and Verify without infringing the RSA Patent ?*

http://www.cyberlaw.com/rsa.html

🕮 GARCÍA DE JALON, JAVIER. RODRÍGUEZ, JOSE IGNACIO. IMAZ, AITOR.

Aprenda Servlets de Java como si estuviera en segundo. Universidad de Navarra, 1999

🕮 GIRO, GRUPO. *Documentación Interna.* Varias fechas.

🕮 OBJECT MANAGEMENT GROUP (OMG). *UML XMI DTD Specification.*

http://www.omg.org

🕮 WORLD WIDE WEB CONSORTIUM (W3C). *Document Object Model (DOM)*

http://www.w3c.org/DOM

Libros

🕮 CAGLE, KURT. *XML Developer's Handbook*. Ed. SYTEX 2000

🕮 HAROLD, ELLIOTE RUSTY. *XML Bible*. Ed. Hungry Minds, 2nd Edition

🕮 JAWORSKI, JAMIE. *Java 1.2 Al Descubierto*. Ed. Prentice-Hall, Madrid 1999

🕮 MENEZES, ALFRED J. van OORSCHOT, PAUL C. VANSTONE SCOTT A. *Handbook of Applied Cryptography*. Ed. CRC 1997.

🕮 NAUGHTON, PATRICK. SCHILDT, HERBERT. *Java 2 : The Complete Reference*. Ed. Osborne/McGraw-Hill

🕮 STINSON, DOUGLAS R. *Cryptography – Theory and Practice*. Ed. Chapman & Hall/CRC. 2nd Edition