Entorno de soporte a la reutilización en Pymes: Un repositorio XML

José L. Barros[1], José M. Marqués[2]

[1]Universidad de Vigo, Departamento de Informática, España
jbarros@uvigo.es
[2]Universidad de Valladolid, Departamento de Informática, España
jmmc@infor.uva.es

Resumen: La mayor parte de empresas dedicadas al desarrollo comercial de software, son del tipo pyme (Pequeñas y Medianas Empresas). Sin embargo, en muchas de ellas no existe una organización adecuada, ni un número de personal suficiente, para imponer el uso de una metodología estricta de ingeniería de software. En este contexto, creemos necesario dotar a las pymes de un entorno de desarrollo flexible, potente, fácil de integrar entre los diferentes roles que juega el reducido personal dedicado al desarrollo, y que pueda adaptarse rápidamente a necesidades cambiantes del mercado. Es en este tipo de entorno donde las técnicas de reutilización, desde la perspectiva de desarrollo-con y –para reutilización, pueden ofrecer un incremento notable en la productividad, calidad del producto, y una reducción importante en los costes. Las pymes presentan pocos problemas de tipo gerencial o administrativo para la implantación de la reutilización, sus principales barreras son de tipo técnico: disponibilidad de componentes reutilizables (repositorio), herramientas de desarrollo potentes (búsqueda, recuperación, adaptación de componentes, ingeniería directa e inversa, etc.) y metodologías flexibles. Este artículo presenta un entorno de desarrollo de software adaptado a estas necesidades, propone un modelo de elemento software reutilizable (ESR) flexible, en formato XML, y se comentan los procesos que permiten transformar cualquier elemento software en ESR, almacenarlo en un repositorio y trabajar con él. En concreto se ofrecen ejemplos de transformación de modelos UML a XML mediante XMI y XSLT, su incorporación al repositorio y los procesos de búsqueda y recuperación posteriores.

Palabras Clave: Elementos Software Reutilizables, intercambio de meta-información, repositorios, XML, XMI.

1 Introducción

Las primeras herramientas para dar soporte al desarrollo de software se centraban en la fase de implementación (programación), editores de texto para escribir el código, y

compiladores de línea de comandos para procesarlo. La siguiente generación proporcionó soporte a otras fases del ciclo de desarrollo, tales como análisis y diseño, mediante herramientas de modelado y, en general, herramientas CASE. Hoy en día los entornos de ayuda al desarrollo, denominados SEE (Software Engineering Environments) cubren un amplio espectro de las fases de desarrollo de software.

Por otra parte, la posibilidad de expresar cualquier elemento software, perteneciente a cualquier fase del ciclo de vida, en un formato único, estándar, facilitaría en gran medida el desarrollo de estos entornos SEE. La integración de herramientas se produciría vía intercambio de información, donde la salida de una herramienta específica podría servir como entrada a otra. En los últimos años los desarrollos en lenguajes de marcado como XML, lenguajes de modelado como UML, y las propuestas de formatos estándar para el intercambio de meta-información como: MOF, CDIF o XMI; posibilitan tal integración [Damm00]. Además, al tratarse de estándares no-propietarios se facilita el desarrollo de herramientas open-source, abaratando costes y logrando independencia del fabricante. La gran aceptación de estas herramientas, y la promoción de ellas que hacen las Universidades y centros de formación, garantizan la presencia en el mercado de profesionales con una capacitación alta.

En cuanto a la reutilización, el elemento central en el acercamiento compositivo, es el repositorio de elementos software reutilizables (ESR). El repositorio debe proporcionar soporte al almacenamiento, clasificación, y posterior recuperación de los ESR. La complejidad en el mantenimiento de un repositorio esta directamente relacionada con la cantidad de ESR que contiene, y con la complejidad intrínseca de los propios ESR, además de involucrar a distintos tipos de usuarios: analistas, programadores, documentalistas, etc. Tal diversidad de usuarios origina un problema de comunicación, una imposibilidad técnica de que todos utilicen y comprendan el mismo lenguaje. En este punto la posibilidad de transformar el formato en que se presenta un ESR, dependiendo del tipo de usuario al que va dirigido, ofrece una solución definitiva al problema de la comunicación. Un analista puede "estudiar" un modelo de alto nivel, expresado en UML, un programador utilizar el código Java o C generado por la herramienta de modelado mediante ingeniería directa, el usuario final podrá revisar la documentación visualizándola como página web en cualquier navegador; mientras que el repositorio solo almacena un documento en formato XML, del cual se extraen todas estas "vistas".

En este artículo presentamos un modelo de repositorio, donde se almacenan ESR expresados como documentos XML, con un modelo "extensible" adaptable a las necesidades de cualquier pyme. El entorno de desarrollo se completa con los procesos de transformación de formatos, y las operaciones de búsqueda y recuperación (Fig. 1). El trabajo se organiza de la siguiente forma: la sección 2 presenta el modelo de repositorio; la sección 3 muestra como llevar a cabo los procesos de transformación de formatos, de un modelo UML, a un documento XMI, y mediante una XSLT [XSLT] transformarlo en una combinación DF+DA expresada en XML; la sección 4 define la similitud funcional entre dos DF almacenadas en el repositorio y muestra el proceso de recuperación basado en cálculo de similitud; La sección 5 ofrece un ejemplo de todo el proceso, abordable por una pyme con pocos recursos; finalmente, la sección 6 presenta las conclusiones.

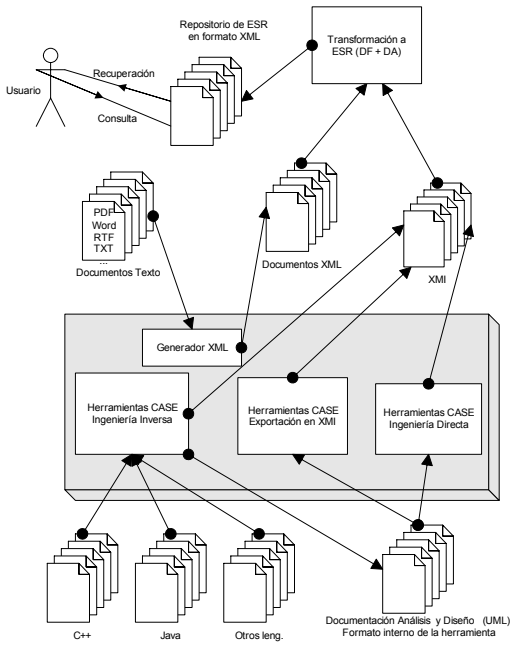

Figura 1 Entorno de desarrollo

2 Modelo de representación: ESR

El repositorio contiene documentos XML, bien correspondientes a modelos UML (expresados en XMI) o a otros subproductos del ciclo de vida de desarrollo. En la actualidad encontramos diversos fabricantes de bases de datos con soporte XML que pueden utilizarse para implementar el repositorio [Marchal04]. Por otro lado, cada asset en el repositorio esta relacionado con un documento XML que agrupa las descripciones funcionales (DF) y administrativas (DA) del asset. Este documento se conoce como descripción de elemento software reutilizable (DESR), y es sobre el cual se ejecutan los procesos de búsqueda y recuperación. Dado que los ESR pueden corresponder a cualquier subproducto del desarrollo de software, es necesario disponer de un modelo que permita representarlos con independencia de la fase del ciclo de vida en que fueron creados. El modelo representa una abstracción de la realidad, a efectos de simplificar el manejo de los ESR y facilitar su comprensión. Los ESR se representan mediante dos grupos diferenciados de descriptores: descripciones funcionales (DF), similares a la propuesta de *Descripciones Software* [Faustle96] [Damiani97b], que representan las funcionalidades ofrecidas por el ESR, en el sentido de métodos, acciones sobre objetos, utilidad, etc.; y la descripción administrativa (DA), que incluye información sobre el tipo de ESR, su autor, la fecha de creación, el coste, los criterios de calidad, etc. Tanto la DF como la DA pueden ser extendidas, para adaptarse a las necesidades particulares de la organización. Al tratarse de

documentos XML podemos ofrecer distintas vistas de la información contenida en estas descripciones, mediante el uso de DTDs adaptadas a los diferentes tipos de usuarios.

Una DF es un conjunto de *características*, que a su vez se componen de un *descriptor* y una *importancia*. El *descriptor* describe la funcionalidad o comportamiento del ESR, mediante un par ordenado que se corresponde con las siguientes alternativas:

1. <verbo, nombre>(verbo expresa acción o función, y nombre representa al objeto sobre el que se realiza la acción); Ej.: <conectar, servidor>
2. <nombre, adjetivo> (donde adjetivo representa alguna característica del ESR); Ej.: <cohesión, alta>
3. <nombre, nombre> (expresa una relación entre dos objetos); Ej.: <socios, listado3>

Mientras que la *importancia* asigna un índice de relevancia a ese *descriptor* dentro de la DF, es decir, la capacidad descriptiva del *descriptor* con respecto al comportamiento general del ESR. Si la DF se corresponde con una consulta del usuario, la *importancia* especifica que tan relevante es ese *descriptor* para el desarrollador. El uso de *importancias* aporta al sistema un mecanismo flexible para aumentar y refinar la capacidad del proceso de recuperación. Los valores de *importancia* se calculan automáticamente , usando una función de pesos adaptada de las funciones clásicas de recuperación de textos.

Resumiendo, definimos:

DF = {característica, .., característica}; donde característica = <descriptor : importancia> y descriptor = <término, término>; finalmente importancia = [Muy poca, Poca, Media, Alta, Muy alta]

Las *características* en una DF pueden ser creadas por el administrador del repositorio (experto, ingeniero de aplicaciones), o pueden ser extraídas automáticamente, o interactivamente, de descripciones textuales del software, tales como: listados de código, páginas de manuales de usuario, encabezados, comentarios, etc. Un protocolo para la extracción automática de características, a partir de código orientado a objeto, ha sido definido en [Damiani97], y a partir de documentación en [Cybulski00]. Actualmente trabajamos en un proceso semi-automático de extracción de características, y por tanto creación de DFs, a partir de documentos XMI, y su transformación vía XSLT.

Un estudio detallado de las DF puede encontrarse en [JISBD02].

Por su parte, la DA esta formada por un conjunto de pares <atributo, valor>, que complementan la información almacenada en la DF. El tipo y la cantidad de pares <atributo, valor> usados en la DA dependerá de las necesidades propias de la pyme. Bajo la premisa de usar XML, la posibilidad de adaptar la estructura de la DA es directa y sencilla. A continuación ofrecemos una indicación de algunos pares

<atributo, valor> que pueden resultar de interés, el primero de ellos, pensamos, es imprescindible, los demás son opcionales:

- ✓ <localización, “url” | string> (* localización del ESR real, bien su URL o un string describiendo su localización física *)
- ✓ <autor, “nombre”>
- ✓ <fecha creación, “fecha”>
- ✓ <fecha última modificación, “fecha”>
- ✓ <fase del ciclo de vida, análisis | diseño | implementación | pruebas>
- ✓ <tipo de ESR, documento | código | patrón | clase | arquitectura | modelo | ... >

El documento XML que expresa la descripción de un elemento software reutilizable (DESR) en función de su DF y DA presenta una estructura similar a la siguiente:

```
<DESR>

<DF>

<term1> término1 </term1>

<term2> término2 </term2>

<imp> importancia </imp>

... (n secuencias term1, term2, imp)

</DF>

<DA>

<localizacion>localización del ESR </localizacion>

<autor> autor </autor>

<creacion> fecha de creacion del ESR </creacion>

...  (otros datos)

</DA>

</DESR>
```

Documento DTD asociado: (*extracto*)

```
<!Element DESR (DF, DA)>

<!Element DF (term1, term2, imp)>
```

```
<!Element DA (localizacion, autor,creacion, ..... , )>
```

Los pares en la DA permiten un tipo de búsquedas más específico que las realizadas sobre las DF, y que pueden resultar de interés en múltiples ocasiones. Por ejemplo, podemos estar interesados en recuperar todos aquellos ESR del repositorio que contengan un modelo UML de la implementación de una GUI; para ello haríamos una búsqueda sobre las DF (funcionalidad, <interfaz gráfica, usuario, *Muy Alta*>) y, sobre los ESR recuperados, una búsqueda sobre las DA (<tipo de ESR, modelo UML>).

2.2. Ejemplos de representación de ESR

A continuación presentamos un ejemplo para clarificar los conceptos anteriores. Solo se ofrece la parte correspondiente a la DF, consideramos que lo relativo a la DA es suficientemente intuitivo.

El ejemplo corresponde a la DF de un componente software para manejar la interface de usuario de una aplicación, usando el sistema X-Windows.

```
{ <conectar,servidor>:M,<definir,ventana>:M,<seleccionar,
eventos>:M,<inicializar,modo gráfico>:P,<mapear, ventana>:M,<manejar,
eventos>:MA,<madurez, alta >:A,
<cohesion,alta>:MA,<documentacion,buena>:P }
```

La distribución de las *importancias* indica que el componente maneja considerablemente bien los eventos enviados por el servidor-X, aunque no es particularmente buena para inicializar el contexto gráfico de la aplicación. Las tres últimas *características* describen la parte no funcional del ESR, *madurez, cohesión y documentación*, mediante el formato <nombre, adjetivo>. El adjetivo "alta" significa que el ESR es "altamente maduro", es decir, estable y bien probado, y la *importancia* "A" indica que este aspecto es "altamente" importante para el reutilizador; mientras que la documentación es buena, pero no es decisiva en la elección de este ESR.

La transformación de esta DF a XML es directa:

```
<DF>

<term1> conectar </term1><term2> servidor </term2>

<imp> M </imp>

<term1> definir </term1><term2> ventana </term2>

<imp> M </imp>

<term1> seleccionar </term1><term2> eventos </term2>

<imp> M </imp>

<term1> inicializar </term1><term2> modo grafico </term2>

<imp> P </imp>
```

```
<term1> mapear </term1><term2> ventana </term2>

<imp> M </imp>

<term1> manejar </term1><term2> eventos </term2>

<imp> MA </imp>

<term1> madurez </term1><term2> alta </term2>

<imp> A  </imp>

<term1> cohesion </term1><term2> alta </term2>

<imp> MA </imp>

<term1> documentacion </term1><term2> buena </term2>

<imp> P </imp>

</DF>
```

3 El proceso de transformación de formatos

Hoy en día podemos encontrar en el mercado numerosas herramientas CASE y de modelado con soporte para XML, y capacidades de importación/exportación XMI. Muchas de estas herramientas están desarrolladas por organizaciones sin fines de lucro y pueden obtenerse gratuitamente desde los sitios web correspondientes. En este trabajo hemos usado dos de ellas: ArgoUML [Argo] [NS00] y MetaMill 3 [Meta], ambos open-source, con capacidades para ingeniería inversa y directa. XMI (XML Metadata Interchange) [XMI] intenta proporcionar una forma estándar de intercambiar cualquier tipo de metadatos que puedan ser expresados usando la especificación MOF (Meta-Object Facility) [MOF] del OMG (Object Management Group) [OMG]. XMI integra tres estándares de la industria: MOF (OMG), UML (OMG), y XML [W3C]. El hecho de que la especificación UML define el meta modelo UML como un meta modelo MOF implica, directamente, que XMI sirve como formato de intercambio para UML. Además, XMI puede usarse para intercambiar información entre almacenes de datos que soporten el meta modelo CWM (Common Warehouse Metamodel, basado en MOF. Por tanto, XMI puede usarse como formato de intercambio entre: repositorios, herramientas de modelado, bases de datos, etc. Puede encontrarse un interesante trabajo sobre la transformación de modelos en [Peltier00], y de generación automática de código a partir de XMI en [Vagsnes02]. Gracias a la incorporación de XML en los principales navegadores, XMI ofrece la posibilidad de compartir información en internet. En concreto, XMI permite a las herramientas case de UML intercambiar sus modelos de datos [Laird01] [Marchal04], y la transformación de documentos como ayuda al desarrollo de aplicaciones [Sarkar01]. Otras características interesantes de XMI son:

1. XMI puede transferir las diferencias entre documentos, de modo que el documento completo solo se transfiere una vez. El nuevo modelo puede obtenerse aplicando al viejo modelo las diferencias.
2. Un documento XMI puede hacer referencia a otro elemento XMI desde otro documento XML usando la tecnología Xlink, y a otra parte dentro del mismo documento mediante Xpath [Jaxen][XPath].
3. XMI proporciona mecanismos de extensión de modo que diferentes herramientas puedan modificar el modelo, sin pérdida de información.
4. XMI soporta la transferencia incompleta de metadatos en el modelo, esto es, XMI puede transferir subconjuntos de un modelo.

Por otro lado, existen diversas herramientas que permiten la transformación de documentación textual, en los formatos más usados (pdf, txt, rtf, word) a documentos XML [Oqbuji03], y cada vez son más los productos de bases de datos con capacidades XML [XMLDB]. Resumiendo, podemos transformar, y reutilizar, cualquier subproducto del desarrollo de software, desde modelos en la fase de análisis, documentación, diseños, código y datos, en un único formato: XML (XMI es un caso particular de XML). Esta posibilidad nos permite intercambiar información entre herramientas, aplicar técnicas de ingeniería directa e inversa, transformar la presentación de cierto elemento software sin modificar el elemento en si, etc. El proceso general puede resumirse en las siguientes tareas:

1. desarrollo del modelo UML, mediante los diagramas necesarios (casos de uso, diagramas de clase, etc.)
2. generación de código mediante ingeniería directa, según las capacidades de la herramienta: Java, C++, etc.
3. exportación de los diagramas UML en formato XMI.
4. transformación de los formatos originales de la documentación (pdf, txt, word, etc.) en XML.
5. generación del documento DESR (DF+DA) en XML, a partir del documento XMI y otra documentación disponible.
6. almacenamiento de todos los documentos XML y XMI en el repositorio.

La Fig.2 presenta las posibilidades de transformación.

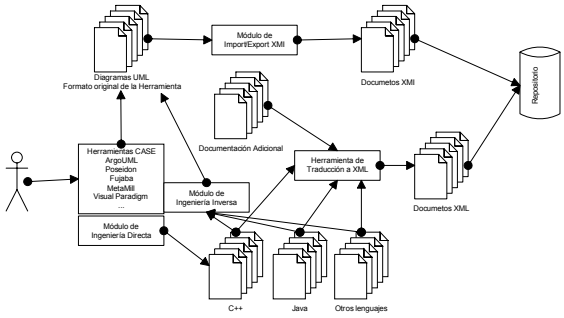

Figura 2 Transformación de formatos

4 La Descripción Funcional: recuperación por similitud

Una vez que disponemos de la DF de un ESR, es necesario un procedimiento que permita valorar cuantitativamente la similitud entre esa DF y una consulta, a efectos de recuperar aquellas DF que más se aproximen a los requerimientos del usuario. Si expresamos la consulta en formato DF, el problema se reduce a comparar dos DF. Tanto para la definición de *similitud*, como para su cálculo, usaremos el acercamiento propuesto en [Toptsis00]. La *similitud* entre DF_1 y DF_2, se define como la probabilidad de que DF_1 pueda ser sustituida por DF_2 mientras se mantienen satisfechos los requisitos de la aplicación en desarrollo. El cálculo se basa en la comparación directa de los *descriptores* en ambas DFs, asignando un grado de similitud según el número y relevancia de los *descriptores* comunes. Por tanto, la similitud puede ser redefinida como una función *Sim*, de los descriptores que son comunes a dos DF's dadas. El valor ofrecido por esta función puede ser expresado por una relación sobre el producto cartesiano de las dos DF, como:

$$Sim : DF \times DF \rightarrow [0,1]$$

El valor *Sim* cumple con las siguientes propiedades:

Intransitiva:

$$SIM(DF1,DF2) > 0 \text{ y } SIM(DF2,DF3) > 0 \rightarrow SIM(DF1,DF3) = ¿?$$

Asimétrica: En general, $SIM(DF1,DF2) \neq SIM(DF2,DF1)$

El procedimiento, de cálculo de similitud, puede implementarse fácilmente en cualquier utilidad software que soporte cálculo matricial, un ejemplo detallado del procedimiento puede encontrarse en [JISBD02].

5 Ejemplo del entorno

A continuación se ofrece un ejemplo sencillo, implementado con herramientas open source que pueden ser fácilmente adoptadas por cualquier pyme, ofreciendo un entorno completo de desarrollo, extensible y configurable. Sin embargo, la diversidad de herramientas disponibles permite una elección amplia, y cada vez es mayor el número de fabricantes y desarrolladores libre que incorporan a sus herramientas capacidades XML.

Para el modelado con UML utilizamos la herramienta ArgoUML, versión 0.14, y construimos um modelo para el alta de un nuevo usuario en el sistema, obteniendo el diagrama de clases siguiente:

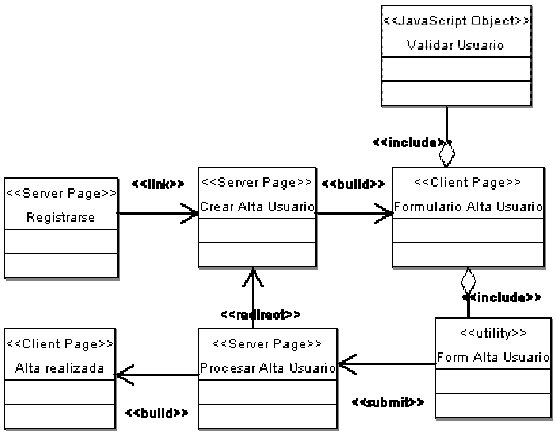

Figura 3 Diagrama de Clases: Alta de Usuario

Al generar la clase "Procesar Alta Usuario" obtenemos el siguiente código Java:

```
public class Procesar Alta Usuario {

    public Form Alta Usuario myForm Alta Usuario;

    public Form Alta Usuario myForm Alta Usuario;

      public Alta realizada <<build>>;

    public Crear Alta Usuario <<redirect>>;

}
```

Un extracto del fichero de intercambio XMI obtenido puede verse a continuación:

```
<XMI.content>

...

<UML:Class xmi.id="xmi.2" name="Registrarse" visibility="public"
isRoot="false" isLeaf="false" isAbstract="false" isActive="false">

<UML:TaggedValue tag="src_lang" value="Java"/>

</UML:Class>


<UML:Class xmi.id="xmi.5" name="Crear Alta Usuario"
visibility="public" isRoot="false" isLeaf="false" isAbstract="false"
isActive="false">

<UML:TaggedValue tag="src_lang" value="Java"/>

</UML:ModelElement.taggedValue>

</UML:Class>

...
```

```
<UML:Class xmi.id="xmi.22" visibility="public" isRoot="false"
isLeaf="false" isAbstract="false" isActive="false"/>

<UML:Generalization xmi.id="xmi.9" subtype="xmi.8" supertype="xmi.22"
visibility="public"/>

<UML:Association xmi.id="xmi.23" name="&lt;&lt;include&gt;&gt;"
visibility="public">

<UML:Association.connection>

<UML:AssociationEnd type="xmi.8" changeable="none"
visibility="public" multiplicity="1" targetScope="instance"
isOrdered="false" isNavigable="true" aggregation="shared"/>

<UML:AssociationEnd type="xmi.14" changeable="none"
visibility="public" targetScope="instance" isOrdered="false"
isNavigable="true"/>

</UML:Association.connection>

</UML:Association>

...
```

El fichero completo del diagrama en formato XMI ocupa 49K, y cada clase java generada automáticamente por la herramienta, 1K. El diagrama tambien puede exportarse en modo gráfico (formatos gif, bmp, wmf o jpg) a efectos de usarlo en herramientas de tratamiento de textos, para la documentación. Una de las críticas más frecuentes a XMI es que es un formato poco legible para el ser humano, aunque muy conveniente para la máquina. Esta situación esta cambiando, y ya existen estudios, y herramientas, para "humanizar" los documentos XMI [Steel01]. Otro pequeño problema es que los formatos XMI usados por las diferentes herramientas CASE pueden variar ligeramente (uso de etiquetas), provocando incompatibilidad, esto puede solucionarse mediante XSLT que transformen el documento XMI salida de una herramienta para adecuarlo al documento XMI que espera como entrada la segunda herramienta [Cooper04].

A partir del documento XMI, del código Java y, posiblemente, de otra documentación adicional, podemos generar automáticamente la descripción del elemento software reutilizable (DF + DA) que sigue (en formato XML):

...(! encabezados XML omitidos)

```
<DF>

<term1>Server Page </term1><term2>Registrarse</term2>

<imp> M </imp>

<term1> Server Page </term1><term2> Crear alta Usuario </term2>

<imp> M </imp>

...

</DF>

<DA>
```

```
<localiz>18622f3:fda9f5d619:-7ffb</localiz>

<autor> nombre del autor </autor>

...

</DA>
```

6 Conclusiones

El entorno de desarrollo de software expuesto en este trabajo permite introducir con facilidad las técnicas de reutilización de software, tanto desde la perspectiva de ingeniería de aplicaciones (desarrollo-con-reutilización) como desde la vertiente generativa (desarrollo-para-reutilización). Los procesos recomendados pueden realizarse con cualquier conjunto de herramientas no propietarias (software libre), lo cual influye directamente en los costos asociados al desarrollo. El modelo de representación de los ESR es fácil de comprender por los distintos tipos de usuarios, y puede ser adaptado con facilidad a las necesidades de una organización particular, o a una situación nueva del mercado. La técnica de clasificación y recuperación es sencilla, y ofrece unos valores de recuperación y precisión muy altos. El previsible desarrollo de UML, XML y las herramientas software asociadas permitirá alcanzar mayores cotas de productividad, calidad y eficiencia, manteniendo el mismo entorno propuesto en este trabajo.

La mayor parte de las metodologías actuales de desarrollo de software, y las herramientas CASE asociadas, ofrecen información suficiente para la construcción / extracción automática de las descripciones funcionales. Muchas empresas que comercializan COTS, ofrecen estas DF como estándar en sus productos, facilitando el único proceso manual (consumidor de tiempo) de nuestra propuesta.

Un repositorio como el propuesto, puede ser implementado sobre sistemas manejadores de bases de datos con soporte XML, y montado sobre un servidor web, con programas cliente de búsqueda y recuperación de ESR, asegurando la escalabilidad del acercamiento. También puede implementarse como una base de datos distribuida, o con interfaces adecuadas para permitir la interoperabilidad con otros repositorios, a través del intercambio de los documentos en formato XML.

7 Referencias

[Argo] http://www.argouml.org/

[Cooper04] Cooper, D., Khoo, B., Konsky, B., Robey, M.: Java Implementation Verification Using Reverse Engineering, 27th Australasian CS Conf., New Zealand, Vol. 26, 2004.

[Cybulski00] Jacob Cybulski; Karl Reed: Requirements Classification and Reuse: Crossing Domain Boundaries. ICSR-6. Austria, June 2000. p.p.: 190-210.

[Damiani97] E. Damiani, M.G. Fugini, C. Bellettini: A Hierarchy-aware approach to faceted classification of object-oriented libraries. Internal Report 001-97. Politecnico di Milano, Dept. Elettronica e Informazione. January, 1997.

[Damiani97b] Damiani, E.; Fugini, M.G.; Fusaschi, E.: A descriptor-based approach to OO code reuse. Computer, Vol. 30, Nº 10 , Oct. 1997. Page(s): 73 –80

[Damm00] Damm C., Hansen K., Thomsen M., Tyrsted M., Tool Integration: Experiences and Issues in Using XMI and Component Technology, In Proc. of TOOLS Europe, 2000.

[Faustle96] S. Faustle; M.G. Fugini: Retrieval of reusable components using functional similarity. Software-Practice and Experience, Vol. 26(5), May 1996. p.p.: 491-530.

[Jaxen] Universal java XPath engine, http://www.jaxen.org

[JISBD02] Barros, J., Marques, J.: Conglomerados Multidimensionales: Un mecanismo simple de organización de Elementos Software Reutilizables. JISBD'02, Madrid, Noviembre, 2002. Pp.: 375-386.

[Laird01] Laird, C.: XMI and UML combine to drive product development. October, 2001. http://www-106.ibm.com/developerworks/xml/library/x-xmi/

[Marchal04] Marchal, B.: Working XML: UML, XMI, and code generation. March, 2004. http://www-106.ibm.com/developerworks/xml/library/x-wxxm23/

[MOF] Meta Object Facility Specification http://www.omg.org/technology/documents/formal/mof.htm

[NS00] Novosoft, Novosoft metadata framework and UML library, http://nsuml.sourceforge.net/

[OMG] The Object Management Group, http://www.omg.org

[Oqbuji03] Oqbuji, U.: An XML format for front office documents, January, 2003. http://www-106.ibm.com/developerworks/xml/library/x-think15/

[Peltier00] Peltier, M.: On levels of model transformation. www.infoloom.com/gcaconfs/WEB/paris2000/S36-02.HTM

[Sarkar01] Sarkar and C. Cleaveland, "Xml based document transformation applied to application software development projects," 2001.

[Steel01] Steel, J.: Generating Human-Usable Textual Notations for Information Models. www.dstc.edu.au/Research/Projects/ Pegamento/publications/edoc01-hutn.pdf

[Toptsis00] Anestis Toptsis: Heuristic Clustering of Reusable Software Libraries. Informatica, Vol. 24, Nº 4, December 2000. p.p.: 487-496.

[Vagsnes02] Vägsnes, K.: Generating code from XMI. November, 2002. fag.grm.hia.no/ikt2340/year2002/ projects/arnewiklund/xmirapport.pdf

[W3C] World Wide Web Consortium (W3C), http://www.w3.org/

[XMI] The XML Metadata Intercahnge (XMI) Specification, http://cgi.omg.org/docs/formal/

[XMLDB] Data Bases with XML support. http://www.rpbourret.com/xml/XMLDatabaseProds

[XPath] Xml path language version 1.0. http://www.w3c.org/TR/xpath

[XSLT] *XSL Transformation(XSLT) Version 1.0*, W3C Std. http://www.w3.org/TR/xslt