Elaboración de aplicaciones software a partir de componentes COTS

Luis Iribarne[1] y Antonio Vallecillo[2]

[1] Dpto. de Lenguajes y Computación. Universidad de Almería
e-mail: liribarne@ual.es
[2] Dpto. de Lenguajes y Ciencias de la Computación. Universidad de Málaga
e-mail: av@lcc.uma.es

Resumen En el área de la Ingeniería del software basada en componentes se está presenciando, cada vez más, el uso de componentes comerciales (COTS) para la elaboración de aplicaciones software. Por este motivo, los desarrolladores de aplicaciones demandan la existencia de mecanismos que les ayuden en las tareas de definición, búsqueda, selección e integración de componentes comerciales. Sin embargo, en la actualidad no existe un criterio unificado para la especificación de componentes COTS, ni tampoco para la especificación de arquitecturas software basadas en este de tipo de componente. Además, los actuales servicios de trading no ofrecen soporte para estos procesos basados en componentes comerciales. En este trabajo presentamos una metodología para la elaboración de aplicaciones software a partir de componentes COTS que se basa en tres aspectos: (a) la descripción de la arquitectura software en UML-RT, (b) un modelo para la especificación de componentes COTS y (c) un proceso de trading para componentes COTS. Esta metodología ha sido implementada con tecnología XML estándar, XMLSchemas, XQuery, CORBA y UML-RT.

Palabras Clave: componentes COTS, arquitecturas de componentes software, trading.

1 Introducción

En el ámbito de la Ingeniería de Software Basada en Componentes (ISBC) se está presenciando un mayor interés por el uso de los componentes comerciales, también conocidos como componentes "off-the-shelf" o componentes COTS [2,4,19]. El uso de esta clase de componente puede tener significativas ventajas, como reducir sus costes, esfuerzos y tiempo de construcción, a la vez que aumenta la fiabilidad y flexibilidad del producto final. No obstante, para sacar provecho de estas ventajas también es necesario que los componentes comerciales cumplan ciertas garantías de fiabilidad relacionadas con aspectos como la calidad de servicio (QoS) que ofrece el componente, o la calidad de la información funcional y no funcional del componente, que luego los arquitectos del sistema software, o incluso procesos automáticos, puedan consultar para comprobar si dicho componente cumple con las exigencias de los componentes especificados en la arquitectura software de la aplicación.

En ISBC existen tres áreas fundamentales para el desarrollo de aplicaciones software basadas en componentes, que son: (a) las arquitecturas software, (b) la documentación y especificación de componentes, y (c) los procesos de trading para componentes. Actualmente, pensamos que estas tres áreas están desligadas entre sí, y no están preparadas para el uso de componentes COTS. Aunque son numerosas las propuestas existentes de lenguajes para la definición de arquitecturas software—como ACME, AESOP o Wright, entre otros—actualmente son pocos los lenguajes apropiados para la descripción de arquitecturas con componentes COTS. Además, tampoco existe ninguna técnica o forma reconocida para documentar y especificar esta clase de componente, y los procesos de trading actuales sólo funcionan para objetos, pero no para componentes comerciales. Por estas razones, debido en parte al rápido crecimiento de las técnicas y tecnologías relacionadas con los componentes comerciales, pensamos que es necesario la existencia de unas pautas o mecanismos que ayuden a los ingenieros en las tareas de construcción de aplicaciones software a partir de componentes COTS.

En el presente trabajo presentamos una metodología que une estas tres áreas de la ISBC—arquitecturas software, documentación de componentes, y procesos de trading—para la elaboración de aplicaciones software a partir de componentes comerciales. La metodología se caracteriza por ofrecer un mecanismo para describir arquitecturas de componentes comerciales a partir de UML-RT. También ofrece una forma de documentar y especificar componentes comerciales, y un proceso de trading para componentes COTS.

El trabajo lo hemos estructurado en siete secciones, la primera la presente introducción. En la sección 2 discutimos algunos aspectos del desarrollo de aplicaciones con componentes comerciales. En la sección 3 presentamos una metodología para la elaboración de este tipo de aplicaciones. En la sección 4 exponemos un experimento de aplicación software hecha a partir de componentes comerciales, para ilustrar la metodología COTS. En la sección 5 comentamos la tecnología utilizada para desarrollar la metodología. En la sección 6, describimos algunos de los trabajos relacionados, y finalizamos con unas conclusiones y trabajo futuro en la sección 7.

2 Aplicaciones basadas en componentes comerciales

Últimamente se está observando una práctica "buy, don't build" en el desarrollo de software, filosofía promulgada por Fred Brooks en 1987 [3] y que abogaba por la utilización de componentes prefabricados, sin tener que desarrollarlos de nuevo. Esto ha permitido procesos de desarrollo *bottom-up*, donde detalles de implementación e implantación de componentes prefabricados son tratados en fases tempranas de la ingeniería de requisitos. Esto ha facilitado el desarrollo de líneas de interés en áreas como las arquitecturas software, la documentación de componentes, y los procesos de trading para componentes COTS. Sin embargo, actualmente estas áreas están desligadas para la elaboración de aplicaciones a partir de componentes comerciales.

Arquitecturas de componentes comerciales. Hoy día son numerosas las propuestas existentes de lenguajes para la definición de arquitecturas software, conocidos simplemente como LDAs. Ejemplos de estos lenguajes son Acme, LEDA, MetaH, Rapide, SADL, UniCon o Wright, entre otros[1]. No obstante, los LDAs actuales carecen de expresividad suficiente para poder desarrollar arquitecturas software que utilicen componentes comerciales, y están limitados para hacer procesos de trading sobre ellos.

En recientes trabajos [8,10,16] hemos observado una tendencia a utilizar notación UML para la descripción de arquitecturas software. Las extensiones de UML, como las restricciones, valores etiquetados, los estereotipos y las notas, permiten representar elementos arquitectónicos como conectores, protocolos o propiedades. Mientras sale la propuesta UML 2.0, hemos adoptado UML-RT [26,27] como propuesta LDA para la metodología. UML-RT es una mezcla entre UML estándar y el LDA de Bran Selic, ROOM (*Real-time Object Oriented Modelling*) [28].

Documentación y especificación de componentes comerciales. Un componente software, en general, puede quedar especificado por medio de sus interfaces, que describen las operaciones, atributos, parámetros y otra información sintáctica. No obstante, esta información sintáctica no es suficiente a la hora de construir aplicaciones [29,30]. También es necesaria la información de los *protocolos*, que describen la interacción del componente, o la información *semántica*, que describe el comportamiento de las operaciones del componente. Para el caso de los protocolos, existen diferentes formalismos para describirlos, como máquinas de estados finitas [29], redes de Petri [1], lógica temporal [14] o el π-cálculo [5]. Para el caso de la información *semántica*, se usan formalismos como pre/post condiciones e invariantes [7], ecuaciones algebraicas [9] o el cálculo de refinamiento [17].

[1] En `http://www.sei.cmu.edu/cbs/index.html` está disponible un estudio sobre LDAs que hizo recientemente el SEI (*Software Engineering Institute*) del Carnegie Mellon University

De forma similar, es necesario un mecanismo para documentación de componentes comerciales, muy útil para tareas de búsqueda, selección y ensamblaje de componentes. Para estas labores, es necesario que la documentación de los componentes contenga información del tipo funcional como la que hemos citado antes—signaturas, protocolos y comportamiento—y también información no funcional y no técnica del componente. En [11] proponemos unas plantillas para documentar componentes comerciales basada en este tipo de información, y en [13] extendemos el trabajo para información no funcional. Como veremos más adelante, estas plantillas las hemos adoptado para nuestra metodología COTS.

Servicio de trading. El servicio de trading, conocido también como función de trading, o simplemente trader, es un objeto software que sirve de intermediario entre unos objetos que ofertan ciertas capacidades, que se denominan servicios, y otros objetos que demandan la utilización dinámica de estas capacidades. La función de trading es una de las 24 funciones del modelo ODP (Reference Model of Open Distributed Processing, RM-ODP), establecida como norma por ISO/ITU-T [23]. Esta especificación ha sido adoptada por el *Object Management Group* (OMG) con el nombre de *CosTrading* para el servicio de trading de *CORBAservices*, y en la actualidad existen diversas implementaciones disponibles en el mercado [20,21,22].

Sin embargo, se ha visto que la actual función de trading de ODP está limitada sólo para objetos, y no es suficiente para el caso de los componentes COTS. En [11] estudiamos las limitaciones del servicio de trading para componentes COTS y presentamos un trader como una extensión de dicho servicio de trading. De nuevo, como veremos más adelante, el trader lo hemos adoptado como parte de nuestra metodología COTS.

3 Metodología basada en COTS

En esta sección describimos una metodología para la elaboración de aplicaciones software basadas en componentes COTS. La metodología se basa en el uso inicial de la arquitectura software AS de la aplicación a construir y de un proceso trader T encargado de localizar componentes que cumplen con las restricciones impuestas en AS.

De forma general, el proceso consiste en enfrentar las especificaciones de los componentes definidos en AS con las especificaciones de los componentes prefabricados residentes en el repositorio del trader. Como resultado del proceso de enfrentamiento, el trader genera un conjunto de combinaciones de componentes que dan solución parcial o total a la arquitectura a construir. Este conjunto de combinaciones es ofrecido al arquitecto de la aplicación, quien puede que modifique algunas restricciones iniciales de la arquitectura en base a las valoraciones que éste haga de las soluciones devueltas por el trader. Puede que el proceso continúe ofreciendo de nuevo al trader la nueva arquitectura y volviéndose a repetir el proceso de trading, así sucesivamente hasta que todas las restricciones software se hayan cumplido o hasta que el propio arquitecto lo decida. En la figura 1 mostramos el proceso completo de la metodología propuesta.

Paso 1. Generación de la arquitectura software *AS*. Primero se definen los requisitos de la aplicación en una arquitectura software AS. Para esto, la metodología propone el uso de UML-RT, pues tiene expresividad suficiente para modelar arquitecturas software. Como muestra la figura 2, UML-RT se basa en un diagrama tradicional "box-and-line". El ejemplo es un conversor de imágenes que utilizaremos más adelante.

En UML-RT los componentes se describen mediante cajas, denominadas *cápsulas*, que pueden contener a su vez otras, y se indica con dos pequeños cuadros conectados (esquina inferior derecha de GTS). Para nuestros propósitos, cada cápsula (componente) puede tener una o más interfaces, denominados *puertos*—los cuadrados pequeños de color negro o blanco—y pueden ser de dos tipos, ofertadas (blanco) y requeridas (negro).

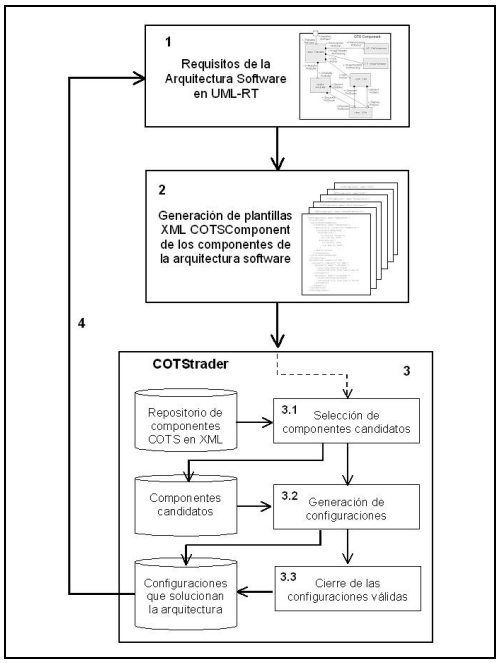

Figura1. Diagrama de la metodología COTS propuesta

En la figura 2, `/sder:Sender` significa que `sder` es una instancia del componente base `Sender`. Cada *puerto* se denota con un nombre y un *protocolo*, que especifica el orden en el que se envían los mensajes. Por ejemplo, un puerto `+transReqSend:ProtTrans` es una interfaz `transReqSend` y un protocolo `ProtTrans`. Por otro lado, `ProtTrans~` es el protocolo dual de la interfaz `transProv`. Por último, los *conectores* son líneas que unen dos puertos. Si un conector une más de un puerto, estos se deben duplicar en la cápsula que lo requiera, mostrado como un doble cuadro pequeño, como sucede por ejemplo en la interfaz `transProv` del componente `GTS`.

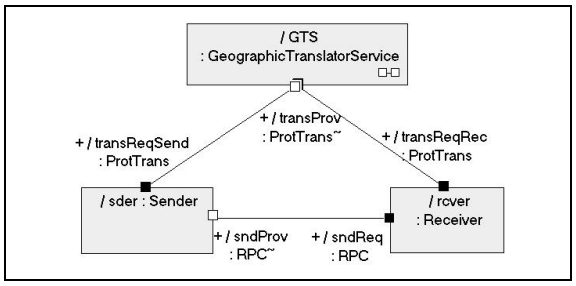

Figura2. Un ejemplo de notación UML-RT

Paso 2. Generación de plantillas COTSComponent. Tras generar la arquitectura software en UML-RT, ahora es necesario extraer los requisitos impuestos para cada componente en la arquitectura software y representarlos en un formato que acepte el proceso de trading (paso 3).

Para este proceso hacemos uso de *COTSComponent*, una plantilla XML para documentar componentes comerciales [11,13]. Como muestra la figura 3, la sección *Functional* describe aspectos funcionales, como descripción de las interfaces ofertadas y requeridas (`ProvidedInterfaces` y `RequiredInterfaces`), pre/post (`behavior`) y protocolos (`serviceAccessProtocol`). La parte *Properties* describe aspectos no funcionales de la forma `(nombre,valor)`. La parte *Packaging* es información de empaquetamiento, implantación e instalación. En el ejemplo, `packaging` está definido en notación `CCM` en un archivo ubicado en la red. La parte *Marketing* recoge información no técnica, como licencia, precio, certificado o detalles del vendedor.

```
<?xml version="1.0"?>
<COTScomponent name="OnePlaceBuffer"
               xmlns="http://www.cotstrader.com/COTS-XMLSchema.xsd"
               xmlns:types="http://www.w3.org/2001/XMLSchema">
 <!-- 1: Functional information -->
 <functional>
  <providedInterfaces>
   <interface name="OnePlaceBuffer">
    <description notation="CORBA-IDL">
     interface OnePlaceBuffer {void write(in long x); long read();};
    </description>
    <behavior notation="Larch"> ... </behavior>
   </interface>
   <interface name="LoginInterface"> ... </interface>
  </providedInterfaces>
  <requiredInterfaces> ... </requiredInterfaces>
  <serviceAccessProtocol> ... </serviceAccessProtocol>
 </functional>
 <!-- 2: Non functional information -->
 <properties notation="W3C">
   <property name="confidentiality">
     <type>xsd:string</type> <value>CRYPTOGRAPHY[PublicKey]</value>
   </property>
   ...
 </properties>
 <!-- 3: Packaging information -->
 <packaging>
   <description notation="CCM-softpkg" href=".../OnePlaceBuffer.csd"/>
 </packaging>
 <!-- 4: Marketing information -->
 <marketing>
   <license href="http:// ... /examples/OPB/license.html"/>
   <expirydate> 05-10-2001 </expirydate>
   <certificate href="http:// .../examples/OPB/lcard.png"/>
   ...
 </marketing>
</COTScomponent>
```

Figura3. Una plantilla para especificar componentes COTS

Paso 3. El proceso COTStrader. El proceso anterior genera una plantilla *COTSComponent* por cada componente definido en UML-RT dentro de *AS*. Estas plantillas son las que el trader utiliza para buscar y seleccionar componentes desde su repositorio. El objetivo final del trader es generar una lista de combinaciones posibles de componentes que den solución a las necesidades impuestas en la arquitectura software de la aplicación.

COTStrader es una herramienta que implementa un servicio de trading para componentes COTS [11]. Como muestra la figura 1 (tarea 3.1), primero se extrae del repositorio aquellos componentes que potencialmente, y de forma individual, podrían formar parte de una solución para *AS*, considerando sólo las interfaces ofertadas. Este proceso genera una lista de componentes candidatos, que es un subconjunto reducido de componentes del repositorio original.

Seguidamente, hemos definido una función que a partir de la lista de componentes candidatos, genera todas las combinaciones posibles de componentes que dan solución parcial o total a las necesidades de *AS* (tarea 3.2 de la figura 1). Por último, se aplica un proceso de cierre a las soluciones para inconsistencias y ajuste a las necesidades de *AS* (tarea 3.2).

Paso 4. Evaluación de resultados. Las condiciones de salida del trader se pueden establecer como políticas internas del trader, o directamente en la petición de consulta. Las soluciones generadas por el trader son devueltas al arquitecto de la aplicación para que éste las evalúe. En este proceso, el arquitecto puede detectar qué componentes de la aplicación habrá que desarrollar, al no existir componentes comerciales para ellos, o cuales de ellos deberá implantar en su sistema. Además, si lo requiere, el arquitecto puede refinar la *AS* inicial para incorporar o eliminar nuevos requisitos (paso 1). De nuevo *AS* se ofrece al trader (pasos 2 y 3) para que busque y genere otras soluciones, ahora en base a las nuevas restricciones. Este recorrido cíclico se repite de forma indefinida hasta que todas las restricciones impuestas en *AS* se hayan cumplido o hasta que el propio arquitecto lo decida.

4 Un experimento de aplicación COTS

Una vez presentada la metodología COTS, en esta sección vamos a desarrollar un experimento de aplicación de componentes COTS para ilustrar el funcionamiento de dicha metodología. El experimento es un servicio de conversión de imágenes espaciales, también conocidas con el nombre de imágenes geográficas (GTS, *Geographic Translator Service*). En la figura 2 ya mostramos el marco del GTS. En este marco, un componente `Sender` tiene que pasarle una imagen con un formato a otro componente `Receiver`. Para ello, `Sender` delega el proceso de conversión al `GTS`, pasándole un mensaje XML con la información para hacer la conversión, por ejemplo de la siguiente forma: `<image url="http://..." inout="River" iF="DWG" oF="DXF" Com="zip"/>`.

Según esto, el `GTS` extrae una imagen `DWG` comprimida en `ZIP` desde el sitio `http`. El `GTS` genera un archivo `DXF` con el mismo nombre, y lo almacena en una celda de un buffer asociado. Cada celda es `<UDDI,href>`, siendo `UDDI` un identificador para la imagen convertida y `href` una referencia hacia la imagen convertida. El `GTS` responde con el `UDDI`, utilizado después para extraer el archivo convertido desde el buffer. Suponemos que el buffer trabaja en XML.

Para no complicar en exceso el ejemplo, sólo vamos desarrollar un subsistema software con el comportamiento interno del componente `GTS`, sin tener en cuenta los componentes `Sender` y `Receiver`. Por tanto, GTS deberá estar formado al menos por un componente compresor de archivos, un componente conversor de imágenes y una API XML.

Paso 1. Definición de la arquitectura software. Para el experimiento usamos Rational Rose RealTime, que adopta UML-RT. En la figura 4 podemos ver la arquitectura de GTS. Aquí, `trans:Translator` es el componente principal con una interfaz `Translator` y otras cuatro requeridas: (a) un compresor de archivos `fC:FileCompressor`, estilo zip/unzip; (b) un conversor de imágenes geográficas `iT:ImageTranslator` como *Geographic Translator* o *MapObjects*; (c) un componente `xDR:XDR` para representación de datos intermedia en XML; (d) un buffer XML `xmlBuf:XMLBuffer` que almacena `<UDDI,href>`; y (e) dos interfaces DOM para trabajar en XML para el XDR y el Buffer, disponibles en paquetes como XML4J de IMB o JAXP de Sun.

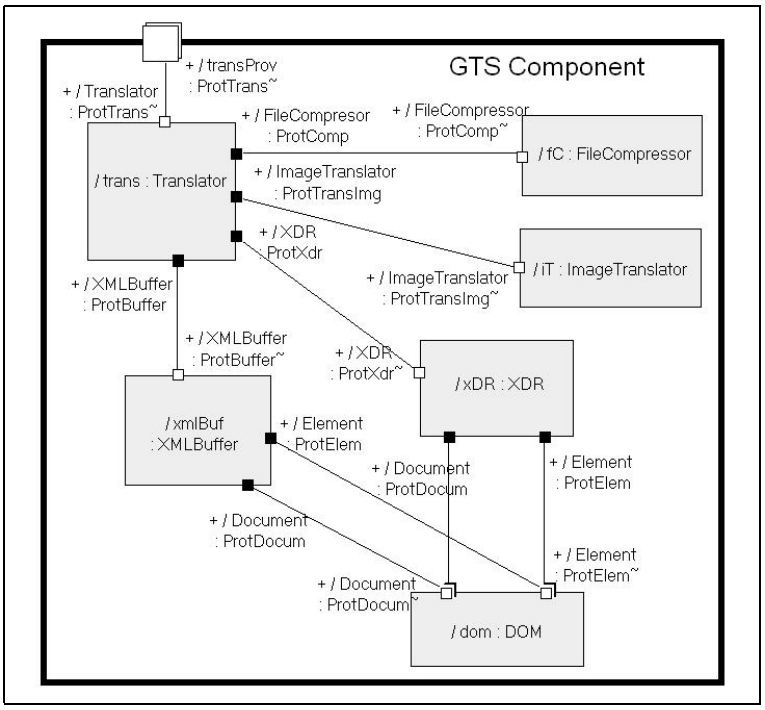

Figura4. La arquitectura software de GTS expresada en UML-RT

Para cada componente, describimos sus requisitos internos como notas UML y valores etiquetados (ver figura 5). Para `DOM` no hemos impuesto ninguna de restricción, sólo que tenga las interfaces `Document` y `Element`. El resto de las cápsulas contienen una nota con la descripción de la interfaz en notación CORBA IDL, esto expresado como un valor etiquetado {`notation=CORBA IDL`}. De forma similar, las propiedades se describen con notas y un valor etiquetado para la notación. Por ejemplo, `ImageTranslator` tiene tres propiedades en OCL en tres notas.

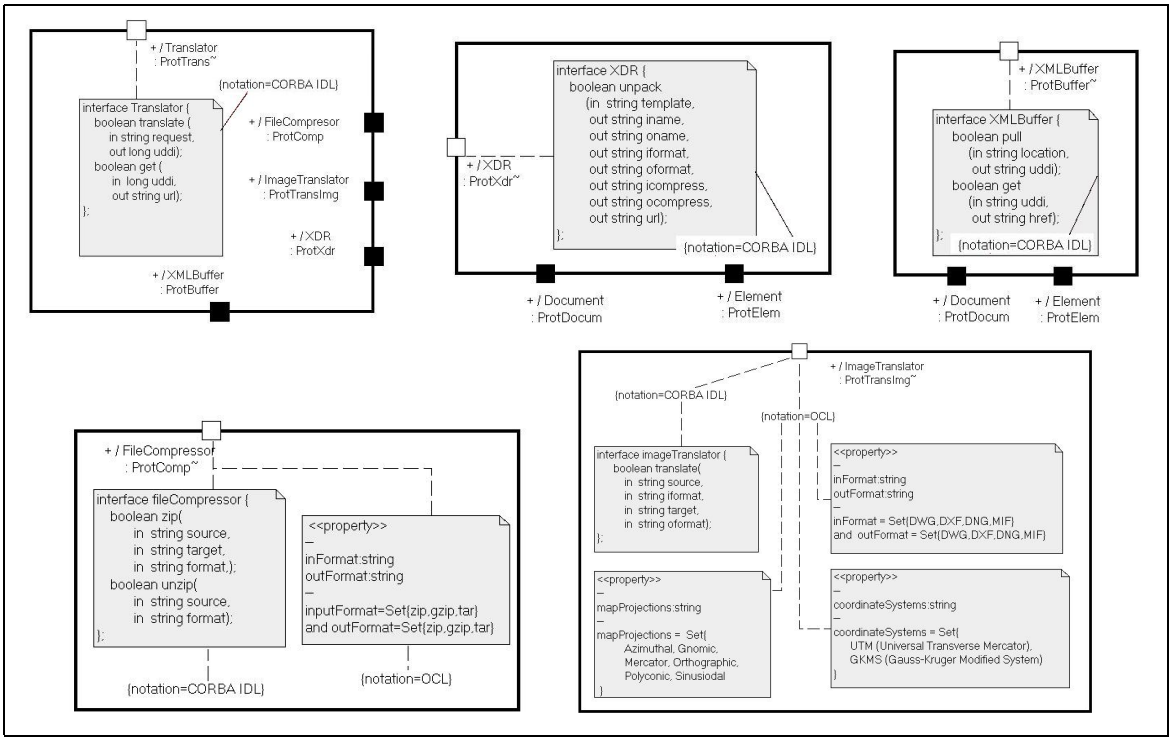

Figura5. Descripción en UML-RT de los componentes del GTS

Paso 2. Generación de plantillas COTSComponent. A continuación se extrae la información de cada cápsula para generar plantillas *COTSComponent*, como muestra la figura 6. En la metodología, la representación gráfica de la arquitectura es almacenada en XMI, que luego se rastrea y se extrae la información necesaria para generar las plantillas XML. Rational Rose RealTime no almacena XMI, sino que lo hace siguiendo un modelo `.mdl` propio de Rational, en modo texto. Estamos desarrollando una herramienta específica para trasladar `.mdl` a XMI.

Paso 3. El proceso COTStrader. Seguidamente las plantillas *COTSComponet* generadas en el proceso anterior, son ofrecidas al trader para que éste busque y seleccione componentes en el repositorio y genere soluciones de composición para las necesidades impuestas en la arquitectura software de GTS. Vamos a suponer que el trader genera sólo las dos siguientes soluciones:

```
S1 = {XML4J:DOM, WINZIP:FileCompressor, MapObject:ImageTranslator}
S2 = {JAXP:DOM, WINZIP:FileCompressor, MapObject:ImageTranslator}
```

Con esta notación, `MapObject:ImageTranslator` significa que se ha encontrado el componente comercial `MapObject` que cumple con las tres propiedades y la interfaz del componente `ImageTranslator`, impuestas cuando desarrollamos *AS* en el paso 1 (ver de nuevo la figura 5). Según esto, las dos soluciones son muy similares, salvo que el trader ha encontrado dos componentes comerciales para `DOM`, `XML4J` de IBM y `JAXP` de Sun.

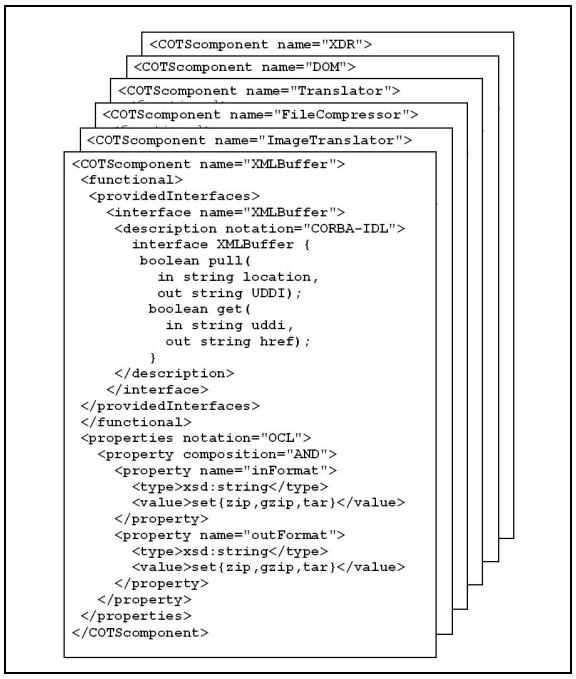

Figura6. Plantillas COTSComponent generadas a partir de la arquitectura software GTS

Paso 4. Evaluación de resultados. Por último, las dos soluciones `S1` y `S2` son ofrecidas al arquitecto de la aplicación para que éste las evalúe. Para estas dos soluciones no se han encontrado componentes comerciales para los componentes `Translator`, `XMLBuffer` y `XDR`. Además, en la dos soluciones hay que decidir por utilizar `XML4J` o `JAXP` para el componente `DOM`. En cualquier caso, el arquitecto podría ahora refinar las restricciones impuestas para estos componentes en la arquitectura software, modificándolas directamente en la descripción UML-RT del paso 1. El proceso de trading se volvería a repetir hasta encontrar una solución con el máximo de componentes comerciales, o hasta que el arquitecto decida finalizar.

5 Tecnología utilizada

Todos los procesos que soportan la metodología propuesta han sido implementados en Java. Para la descripción de la arquitectura software, hemos utilizado la herramienta Rational Rose RealTime, que adopta la versión original UML-RT de Bran Selic. Para escribir las plantillas *COTSComponent* hemos utilizado notación XML 1.0 del W3C. Para desarrollar los esquemas de estas plantillas hemos utilizado la versión 1.1 de XMLSchema, también del W3C. Para la implementación del trader hemos utilizado el ORB Orbacus de IONA. Para el repositorio de plantillas de componente y para la definición de las propias plantillas, hemos utilizado la API XML4J de IBM. También hemos implementado un cliente web para dar servicio de trading por Internet desde `http://www.cotstrader.com`, el sitio web de la propuesta metodológica. Para hacer la implementación del sitio web y acceso al servicio de trading, hemos implementado varias Servlets para el Servidor Web Tomcat Apache, bajo Linux/RedHat 7.2.

Finalmente, para ciertas partes de la metodología propuesta también hemos hecho uso puntual de formalizaciones conocidas. Por ejemplo, hemos utilizado OCL para la descripción de las propiedades de una plantilla COTS. Para la especificación semántica de las interfaces de un componente COTS—indicadas dentro de la etiqueta `behavior` de una plantilla *COTSComponent*, véase la figura 3—hemos utilizado la notación JML de Leavens [15]. Para la descripción de protocolos—indicado dentro de la etiqueta `serviceAccessProtocol` de una plantilla *COTSComponent*, véase de nuevo la figura 3—hemos utilizado notación π-calculo [18].

6 Trabajos relacionados

Las aportaciones realizadas en este trabajo están relacionadas con varias líneas de investigación. Una de ellas tiene que ver con los procesos de selección y búsqueda en repositorios software. Aquí destacamos el trabajo de Rolland [24], que propone una técnica para la adquisición de requisitos capturados mediante diagramas de transición basados en modelos As-Is, To-Be, COTS, y de emparejamiento. Sin embargo, la propuesta es algo abstracta, y no propone ninguna forma particular de especificar y buscar componentes COTS en un repositorio. Otra propuesta interesante es la que hace Goguen [9], en la que se presenta una serie de criterios para la búsqueda y selección de componentes en repositorios. Sin embargo, esta propuesta tan solo trabaja para componentes con interfaces simples, y no para componentes con múltiples interfaces ofertadas y requeridas, tal y como nosotros especificamos los componentes en nuestra metodología.

Otra línea de investigación tiene que ver con los trabajos en construcción de aplicaciones con componentes comerciales. En este caso destacamos dos trabajos. En primer lugar destacamos en trabajo Robert Seacord et al. [25] que propone unos procesos para la identificación de componentes basada en conocimiento de las reglas de integración del sistema. Aunque, la propuesta carece de una forma concreta para documentar componentes comerciales, es una de los pocos trabajos que tratan con ejemplos reales de componentes comerciales. Este trabajo se desarrolla dentro del *COTS-Based Systems (CBS) Initiative* del *Software Engineering Institute* (SEI).

En segundo y último lugar encontramos la propuesta de los proyectos CAFE y ESAPS[2], ambos del *European Software Institute* (ESI). Aunque son varias las líneas de interés, aunque destacamos la línea de Cherki et al. [6], que describen una plataforma llamada Thales para la construcción de sistemas software basados en partes COTS. Esta propuesta utiliza herramientas Rational para definir la arquitectura software, aunque utiliza un diagrama de clases, en lugar de UML-RT como proponemos en la metodología aquí propuesta. Además, el trabajo carece de procesos de trading para componentes COTS.

7 Conclusiones y trabajo futuro

En el presente trabajo hemos propuesto una metodología para la elaboración de aplicaciones software mediante ensamblaje de componentes COTS. Esta propuesta se debe en parte a la necesidad de unir tres áreas de la Ingeniería del Software basada en componentes COTS que actualmente se encuentran desligadas, y que son: las arquitecturas software, la documentación y especificación de componentes, y los procesos de trading.

Para el caso de las arquitecturas software, y a falta de una propuesta adecuada para componentes comerciales, pensamos que UML-RT es un mecanismo adecuado como lenguaje para la descripción de arquitecturas software con componentes COTS. En este trabajo extendemos las cápsulas de UML-RT para describir la información de un componente mediante notas y valores etiquetados. Para el caso de la documentación de componentes, proponemos el uso de *COTS-Component*, unas plantillas XML para la especificación de componentes comerciales [11,13]. Por último, hemos visto que los actuales procesos de trading no son adecuados para el caso de componentes COTS y no existe ninguna conexión con las arquitecturas software. Para la metodología usamos *COTStrader* [11], una herramienta que extiende el servicio de trading de ODP para buscar componentes COTS en un repositorio, y una función de composición [12], que genera soluciones a partir de los componentes que encuentra el trader.

Como línea de trabajo futuro estamos interesados en establecer métricas y heurísticas para que el trader pueda generar secuencias de ordenación en base a ciertos criterios establecidos por el usuario o el administrador del trader. También se debe permitir la conexión con otros traders para efectuar operaciones de búsqueda o registro de componentes de forma federada.

[2] Disponibles en `http://www.esi.es/Cafe` y `http://www.esi.es/esaps`

Referencias

1. R. Bastide, O. Sy, and P. Palanque. Formal Specification and Prototyping of CORBA Systems. In *ECOOP'99*, number 1628 in LNCS, pages 474–494. Springer-Verlag, 1999.
2. P. Brereton, D. Budgen, K. Bennet, M. Munro, P. Layzell, L. MaCaulay, D. Griffiths, and C. Stannett. The Future of Software. *Communications of the ACM*, 42(12):78–84, Dec. 1999.
3. F. Brooks. No Silver Bullet: Essence and Accidents of Software Engineering. *IEEE Computer*, 20(4):10–19, Apr 1987.
4. A. W. Brown and K. C. Wallnau. The Current State of CBSE. *IEEE Software*, 15(5):37–46, Sep./Oct. 1999.
5. C. Canal, L. Fuentes, J. M. Troya, and A. Vallecillo. Extending CORBA Interfaces with π-calculus for Protocol Compatibility. In *TOOLS'00*, France, June 2000. IEEE Computer Society Press.
6. S. Cherki, E. Kaim, N. Farcet, S. Salicki, and D. Exertier. Development Support prototype for system families based on COTS. Technical report, ESAPS Project, 2001. http://www.esi.es/esaps.
7. K. K. Dhara and G. T. Leavens. Forcing Behavioral Subtyping Through Specification Inheritance. In *18th International Conference on Software Engineering (ICSE-18)*, pages 258–267. IEEE Press, 1996.
8. D. Garlan, S. Cheng, and A. Kompanek. Reconciling the Needs of Architectural Description with Object-Modeling Notations. *Science of Computer of Computer Programming Journal. Elsevier Science*, 2001.
9. J. Goguen, D.Ñguyen, J. Meseguer, Luqi, D. Zhang, and V. Berzins. Software Component Search. *Journal of Systems Integration*, 6:93–134, Sept. 1996.
10. C. Hofmeister, R. L. Nord, and D. Soni. Describing Software Architecture with UML. In *First Working IFIP Conference on Software Architecture*. Kluwer Academic, 1999.
11. L. Iribarne, J. M. Troya, and A. Vallecillo. Trading for COTS Components in Open Environments. In *27th Euromicro Conference*, Warsaw, Poland, Sept. 2001. IEEE Computer Society Press.
12. L. Iribarne, J. M. Troya, and A. Vallecillo. Selecting Software Components with Multiple Interfaces. In *28th Euromicro Conference*. IEEE Computer Society Press, Sept. 2002. En prensa.
13. L. Iribarne, A. Vallecillo, C. Alves, and J. Castro. A Non-Functional Approach for COTS Components Trading. In *Workshop on Requirements Engineering. Buenos Aires, Argentina*, Nov. 2001.
14. D. Lea. Interface-Based Protocol Specification of Open Systems using PSL. In *ECOOP'95*, number 1241 in LNCS. Springer-Verlag, 1995.
15. G. T. Leavens, L. Baker, and C. Ruby. *Behavioral Specifications of Businesses and Systems*, chapter JML: A Notation for Detail Desing. Kluwer Academic, 1999. http://www.cs.iastate.edu/ leavens/JML.html.
16. N. Medvidovic, D. S. Rosenblum, J. E. Robbins, and D. F. Redmiles. Modeling Software Architectures in the Unified Modeling Lenguage. *ACM Trans. on Software Engineering and Methodology*, 11(1):2–57, Jan. 2002.
17. A. Mikhajlova. *Ensuring Correctness of Object and Component Systems*. PhD thesis, Åbo Akademi University, October 1999.
18. R. Milner. The Polyadic π-calculus: A Tutorial. *Logic and Algebra of Specification. Springer-Verlag*, 1993.
19. C.Ñcube and N. Maiden. COTS software selection: The need to make tradeoffs between system requirements, architectures and COTS components. In *Continuing Collaborations Successful COTS Development*, 2000.
20. OOC. ORBacus Trader. ORBacus for C++ and Java. Technical report, Object Oriented Concepts, Inc., 2000. http://www.ooc.com/ob.
21. OpenORB. The OpenORB web site. Distributed Object Group, 2001. http://www.multimania.com/dogweb.
22. PrismTech. Trading Service - White Paper. PrismTech OpenFusion, Enterprise Integration Services, January 2001. http://www.prismtecnologies.com.
23. RM-ODP. Reference Model for Open Distributed Processing. Rec. ISO/IEC 10746-1 to 10746-4, ITU-T X.901 to X.904, ISO/ITU-T, 1997.
24. C. Rolland. Requirements Engineering for COTS Based Systems. *Elsevier, Information and Software Technology*, 41(14):985–990, 1999.
25. R. C. Seacord, D. Mundie, and S. Boonsiri. K-BACEE: Knowledge-Based Automated Component Ensemble Evaluation. In *27th Euromicro Conference*, Warsaw, Poland, Sept. 2001. IEEE Computer Society Press.
26. B. Selic. UML-RT: A Profile for Modeling Complex Real-Time Architectures. Technical report, Object Time Limited, 1999.
27. B. Selic. On Modeling Architectural Structures with UML. In *UML'2001 Workshop on Software Architectures and Requirements Engineering (STRAW'01)*, Toronto, Canada, May 2001.
28. B. Selic, G. Gullekson, and P. Ward. *Real-Time Object-Oriented Modeling*. John Wiley&Sons, 1994.
29. D. M. Yellin and R. E. Strom. Protocol Specifications and Component Adaptors. *ACM Transactions on Programming Languages and Systems*, 19(2):292–333, Mar. 1997.
30. A. M. Zaremski and J. M. Wing. Specification Matching of Software Components. *ACM Trans. on Software Engineering and Methodology*, 6(4):333–369, Oct. 1997.