Desarrollo de Sistemas Basados en Componentes Utilizando Diagramas de Secuencia.

Miguel A. Pérez Toledano[1], Amparo Navasa Martínez[1],
Carlos Canal[2], Juan M. Murillo Rodríguez[1]
[1]Universidad de Extremadura
Departamento de Informática, Grupo Quercus de Ingeniería del Software
{toledano, amparonm, juanmamu}@unex.es
[2]Universidad de Málaga
Departamento de Lenguajes y Sistemas Informáticos, Grupo GISUM
canal@lcc.uma.es

Resumen

En el mundo actual cada vez se exigen soluciones software de mayor calidad y con menores costes y tiempos de desarrollo. En este contexto, la reutilización de componentes software es una línea de trabajo prioritaria. No obstante, la construcción de software utilizando componentes no puede limitarse a seleccionar componentes que sean acordes con las necesidades del sistema desde un punto de vista exclusivamente sintáctico –en cuanto a los nombres y parámetros de los servicios que ofrecen– y semántico –en cuanto a la función que realizan dichos servicios. Es necesario asegurar además la compatibilidad de los protocolos de interacción de cada componente con los del sistema en que va a ser integrado. Este trabajo propone el uso sistemático de los diagramas de secuencia de UML como herramienta para la descripción de los protocolos de interacción de los componentes de un sistema software. Los diagramas de secuencia se pueden utilizar a distintos niveles de abstracción, agrupando o separando componentes en función de las necesidades, obteniendo así protocolos para componentes individuales o para agrupaciones de componentes. Estos protocolos, especificados mediante grafos y álgebras de procesos, se pueden guardar en repositorios junto a la restante información sobre el componente, y permiten estudiar su compatibilidad con el entorno donde va a ser integrado.

Abstract

In the world we live, software solutions are more and more demanded. These solutions must be not only of excellent quality, but their development costs and time must be as minimal as possible. In this context, component reuse, is an essential research line. However, software development using components cannot be restricted to the selection of components syntactic and semantically compatible with system requirements. In addition, it is necessary to ensure the compatibility of the interaction protocols of each component with those of the system in which it will be integrated. In this paper we propose the use of UML sequence charts for describing the interaction protocols of software components. Sequence charts can be used at different abstraction levels, grouping or dividing components according to the requirements, and obtaining protocols for individual components or for groups of them. These protocols, specified by means of graphs and process algebras, can be stored into repositories, together with the rest of the information about the component. Thus, it will be possible to analyze the compatibility with the environment in which they are going to be placed.

1. Introducción

Los rápidos cambios producidos en los sistemas de información de las organizaciones han disminuido el tiempo y los recursos asignados para la realización de proyectos software. Para adaptarse a estos nuevos retos, la Ingeniería del Software ha evolucionado buscando la generación de sistemas software en los cuales la rapidez en el desarrollo no hipotecase la calidad de las soluciones creadas. En este contexto, la reutilización se ha perfilado como línea de trabajo prioritaria. No obstante, la creación de sistemas usando componentes previamente desarrollados y probados,

no garantiza por sí sola la calidad de los sistemas resultantes, sino que existen diversos problemas de integración [GAO95, GB98] que hay que solucionar para integrar adecuadamente los componentes.

La composición de sistemas reutilizando software implica la búsqueda y selección de componentes que proporcionen los servicios requeridos dentro del sistema [MMM94, PB96, PP93]. Además, hay que establecer una verificación sintáctica que permita comprobar si la sintaxis de dichos servicios es congruente entre los diferentes componentes [ZW95]. Pero con eso no basta, existen problemas de integración, como los bloqueos, que son consecuencia de la incompatibilidad de los elementos integrados. Estudiar la compatibilidad de un componente dentro de un sistema, requiere comprobar si el comportamiento del componente es coherente con el comportamiento del entorno donde va a ser integrado. Nuestra propuesta se basa en documentar los protocolos de interacción entre los componentes que intervienen en un sistema, de manera que estos protocolos puedan ser utilizados para realizar operaciones de verificación durante el diseño del sistema.

Existen diferentes enfoques para modelar y verificar protocolos, que van desde el uso de lenguajes formales específicos [BHS91, T93], o de diversos tipos de formalismos, tales como máquinas de estados [BZ83, B78], redes de Petri [BT82, D82], álgebras de procesos [YS97, C01], lógica temporal [J00], o contratos de reuso [S96]. Sin embargo, la representación de las interacciones entre componentes usando estas técnicas no permite la comprensión intuitiva de las especificaciones obtenidas. Además muchas de ellas requieren al desarrollador el conocimiento de herramientas de especificación complejas. Este trabajo propone aprovechar los diagramas de secuencia de UML ([UML]) para diseñar los protocolos de interacción de los componentes que intervienen en el sistema. Los diagramas de secuencia son una herramienta gráfica fácil de usar, que genera especificaciones sencillas de comprender y que además están integrados dentro de un lenguaje de modelado ampliamente aceptado.

El objetivo de este trabajo no consiste únicamente en diseñar protocolos, se pretende con ellos completar la información de los interfaces [CPT01], obteniendo patrones de interacción adecuados a cada uno de los componentes del sistema. Estos patrones de interacción son utilizados metodológicamente para realizar operaciones de verificación durante la construcción y mantenimiento de sistemas basados en componentes.

Para obtener los protocolos, se describen todos los diagramas de secuencia necesarios para representar las interacciones entre los componentes que interviene en el sistema descrito. Una vez terminada esta tarea, se obtienen las proyecciones de cada componente (las interacciones en que interviene cada uno de ellos) y se organiza la información obtenida. Esta información se sintetiza mediante grafos etiquetados cíclicos orientados (máquinas de estados extendidas) y descripciones algebraicas. Cada una de estas descripciones proporciona una utilidad diferente. Por una parte, la construcción de grafos facilita la detección de inconsistencias en las especificaciones, a la vez que permite representar toda la metainformación aportada por los diagramas (invariantes, regiones críticas, variables...). No obstante, para realizar la verificación de la compatibilidad de protocolos, no es necesario utilizar toda la información sintetizada en los grafos y es más sencillo verificar usando un álgebra de procesos. Por esta razón, es conveniente realizar descripciones algebraicas para cada uno de los componentes que han sido descritos.

La estructura de este trabajo es la siguiente: en la Sección 2 se explican brevemente la propuesta presentada; en la Sección 3 se explica cómo usar los diagramas de secuencia para obtener los protocolos de interacción de los componentes; la Sección 4 presenta cómo usar estos protocolos para el análisis de diversas propiedades; en la Sección 5 se exponen las conclusiones; y por último se presenta la bibliografía utilizada.

2. Tratamiento metodológico de los diagramas de secuencia.

En este apartado vamos a explicar como usar los diagramas de secuencias para facilitar la construcción de sistemas software basados en componentes. Los pasos a realizar son los siguientes:

- La primera tarea consiste en modelar el sistema utilizando las herramientas proporcionadas por UML. En este apartado es interesante centrarse en los diagramas de secuencias, los cuales nos permitirán describir los componentes candidatos que intervienen en el sistema, así como en las interacciones existentes entre ellos.

Los diagramas de secuencia fueron inicialmente pensados para refinar los requisitos de usuario, y profundizar en el conocimiento del sistema a construir, pero dan una visión parcial del sistema centrándose, a menudo, en intentar comprender un caso de uso describiendo ejemplos. En este trabajo se plantea la idea de usarlos, para sintetizar a partir de ellos el comportamiento de cada componente del sistema. Para ello, una vez realizada la descripción de requisitos, utilizando diagramas de casos de uso, es necesario diseñar correctamente el conjunto de escenarios que describen el comportamiento de cada caso de uso. Una vez especificados los escenarios y completada la información con los diagramas de componentes de UML, estarán identificados todos los componentes que intervienen y las interacciones entre ellos.

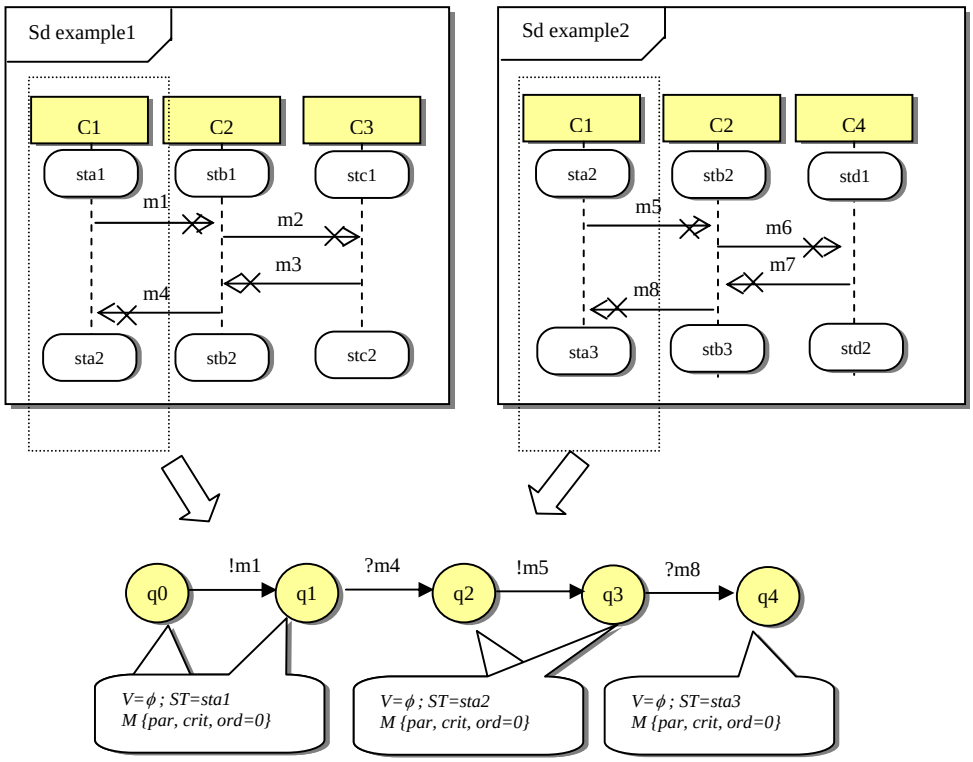

Figura 1: Diagramas de secuencia y grafo de interacción de C1.

- El segundo paso consiste en obtener el comportamiento individual esperado de cada componente. Para ello habrá que seleccionar los escenarios en los que participa y obtener el comportamiento expresado en cada uno de ellos. Para poder relacionar estos comportamientos parciales el trabajo propone la utilización de etiquetas de condición, las cuales serán usadas en los diagramas de secuencia, al principio y final de la línea de vida de cada componente. La síntesis de las informaciones proporcionadas permitirá obtener el comportamiento de cada componente dentro del sistema (protocolo de interacción). Para comprender mejor lo explicado supongamos que hemos especificado un sistema mediante dos diagramas de secuencias (figura 1). Para obtener el protocolo de interacción del componente C1, hay que obtener las proyecciones del componente y organizar la información obtenida. La síntesis de esta información permite conseguir grafos que expresan las interacciones del componente. El uso que podemos darle a estos grafos se describe en la sección 4 de este trabajo.

- Una vez especificados los comportamientos esperados de cada componente el siguiente paso consiste en construir el sistema. Para ello hay que obtener los componentes especificados, de repositorios o implementándolos, e integrarlos dentro del sistema. La especificación desarrollada durante el modelado,

permite describir los servicios requeridos y la sintaxis precisa de cada componente, pero además añade la información sintetizada de los diagramas de secuencia. La especificación de cada componente será usada para realizar las operaciones de búsqueda y selección, la novedad consiste en utilizar los protocolos obtenidos de cada componente para verificar si los elementos seleccionados son compatibles con el entorno en el que han de integrarse. Para facilitar la tarea de selección, se propone la creación de agrupaciones de componentes para facilitar la búsqueda de candidatos. La especificación de estas agrupaciones de componentes se realizará utilizando las especificaciones individuales de los componentes que la forman. Para la creación del protocolo de la agrupación, los diagramas de secuencia descritos durante el modelado permiten construir un protocolo que elimine todas las interacciones entre los componentes agrupados centrándose en las interacciones de estos con el entorno que les rodea.

3. Obtención de protocolos de interacción a partir de los diagramas de secuencia.

En el apartado anterior se ha comentado la manera de desarrollar sistemas basados en componentes, centrando el estudio en documentar las interacciones de los componentes. En este punto expondremos que información incluir dentro de los diagramas de secuencia y como tratarlos para obtener los protocolos de interacciones de los componentes que intervienen en le sistema.

Como se ha comentado anteriormente, los diagramas de secuencias dan solamente una visión parcial del sistema que se construye, de manera que hay que relacionar varios de ellos para tener una idea global del comportamiento de un participante dentro del sistema. Para resolver este problema se pueden utilizar los diagramas de visión general (*Overview Diagrams*) de UML. Estos permiten describir como se relacionan los diferentes diagramas de interacción descritos, pero están enfocados a describir el funcionamiento global de todo el sistema. Sin embargo, nuestro interés se centra en describir el comportamiento individual de cada participante, y comprobar su evolución dentro del sistema software. Para conseguirlo, se propone la utilización de etiquetas de condición de UML. Estas etiquetas se incluirán, como mínimo, al principio y final de la línea de vida[1] de cada participante. Las ventajas del uso de estas etiquetas son las siguientes:

- Definen el estado en que se encuentra un participante al empezar y al finalizar un escenario. Esto permitirá construir fácilmente los Diagramas de Estados de cada participante, eliminando las discontinuidades entre diagramas.

- Permiten detectar inconsistencias en las especificaciones, ya que la evolución de un participante podrá ser estudiada en función de los estados por los que ha pasado. De esta manera, es posible descubrir incongruencias en los estados descritos así como estados inaccesibles.

    - No se necesita tener una visión global del sistema para conocer la evolución de cada participante.

De entre toda la colección de diagramas de secuencia utilizados para describir el sistema, es necesario seleccionar aquellos en los cuales interviene el participante que está siendo estudiado, y de cada uno de ellos se obtiene la traza que describe su flujo de mensajes. Por lo tanto, existirá para un mismo componente, toda una colección de trazas que hay que coordinar. Todos los componentes que intervienen en un sistema evolucionan desde un estado inicial, y van hilvanando una secuencia de eventos que van determinando su comportamiento. En estas condiciones, hay que construir el conjunto de interacciones de un participante en función de las secuencias parciales descritas, cuya unión servirá para comprender la evolución experimentada. Para ello, de entre todas las etiquetas iniciales hay que seleccionar una que represente el estado inicial del participante y a partir de ahí se irán concatenando la ejecución de las demás trazas. El problema se plantea cuando no se pueden unir todas las trazas. En este caso, se trata de un

[1] Se entiende por línea de vida de un participante la proyección de sus interacciones dentro de un diagrama de secuencias.

problema de especificación de requisitos, causado porque faltan escenarios por describir o porque existe un problema en la descripción del estado de inicio o final de alguna de las trazas. Por último, en la descripción de escenarios es frecuente, que se solapen descripciones de escenarios. Para arreglar estas redundancias, a la descripción final se le aplican unos sencillos algoritmos que las eliminan del diseño creado.

De la misma manera que los diagramas de secuencias se han organizado para expresar el comportamiento de un participante, también se pueden usar para expresar propiedades o comportamientos erróneos dentro del sistema. Los protocolos parciales obtenidos de estos diagramas se tratarán de manera diferente a los anteriores y únicamente servirán para realizar operaciones de verificación para chequear ese tipo de situaciones

3.1 Organización de trazas y síntesis de grafos

Para ilustrar lo explicado, vamos a realizar un sencillo ejemplo que permite entender mejor la propuesta. Supongamos que se han especificado los diagramas de secuencia de un sistema, en el cual interviene el componente llamado "I", de los diagramas se han extraído 5 trazas que explican la evolución positiva de un participante estudiado (Figura 2).

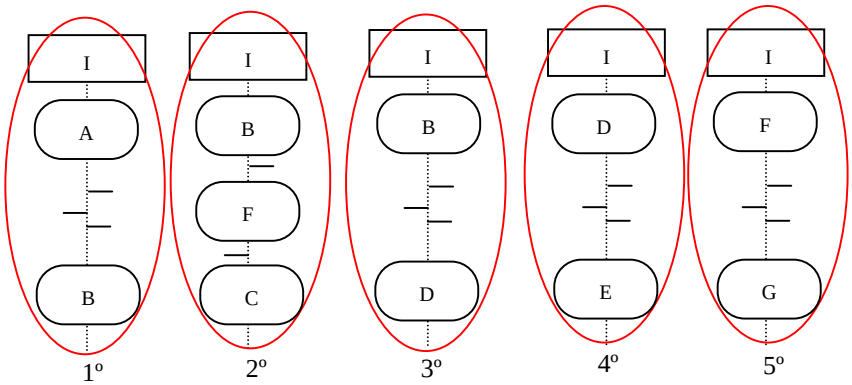

Figura 2: Conjunto de trazas del componente "I".

Una vez seleccionadas las trazas, la siguiente tarea será comprobar la congruencia de las especificaciones. Hay que elegir la etiqueta de inicio (en el ejemplo la etiqueta "A"). La información sobre la etiqueta que especifica el estado inicial se obtiene a partir del conocimiento que el desarrollador tiene sobre el comportamiento del componente. A partir de ella, hay que comprobar que el comportamiento expresado en los diagramas de secuencia es completo y describe la evolución total del participante. Para ello, se estudia la evolución del participante a partir de la concatenación de los comportamientos dispersos por los diferentes escenarios. Esta tarea que se realiza automáticamente, consiste en comprobar que todos los escenarios descritos están unidos describiendo un comportamiento continuo. La detección de discontinuidades o estados inaccesibles implica la existencia de errores en las especificaciones. En la figura 3 puede observarse que todas las etiquetas están unidas y pueden ser accesibles a partir de la etiqueta inicial "A", lo cual demuestra que existe coherencia en el comportamiento descrito, pero no garantiza la completitud de las especificaciones. Si se hubiesen detectado errores en las especificaciones, estos pueden ser debidos a errores en los escenarios descritos o a la falta de escenarios que completen a los existentes. Estos errores deben ser corregidos antes de crear los protocolos de cada componente.

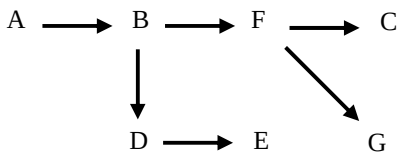

Figura 3. Relación entre las etiquetas de estado usadas en los diagramas de secuencia.

Los diagramas de secuencia proporcionados por UML constituyen una herramienta de modelado muy potente que no se limita a describir el paso de mensajes entre participantes en un sistema, sino que incluyen una abundante y variada información. Para sintetizar la información de las trazas, nuestra propuesta consiste en utilizar grafos etiquetados cíclicos y orientados (máquina de estados extendida). Estos grafos, permiten sintetizar toda la información de los diagramas de secuencias, dando una visión completa del comportamiento del elemento software que está siendo descrito.

3.2 Grafos de interacción.

En esta sección se define el grafo obtenidos en la síntesis de los diagramas, que como se ha comentado anteriormente, es una máquina de estados extendida. Consiste en un grafo cíclico orientado, con etiquetas en las aristas e información en los vértices. Las etiquetas permiten representar el envío o recepción de mensajes, restricciones de ejecución (obtenidas al sintetizar fragmentos tipo *alt, opt* y *loop*) y contadores (usados para representar iteraciones). Por otra parte, cada vértice es una tupla que contiene información acerca del estado de la transición que es descrita.

Definición 1. Un grafo de interacciones es un grafo de la forma (V, E, I, Condición, Etiqueta, Acción, Vert-Inicial, Conj_Vert_Finales) tal que:

- V es un conjunto de vértices.
- E es el conjunto de aristas.
- I es una relación, que asocia a cada arista e $\in$ E con dos vértices <u,v>$\in$ V, llamados los extremos, tal que u=origen(e) y v=destino (e).
- Condición: E $\rightarrow$ CondArista es una función inyectiva que asocia una condición a cada arista del grafo, donde CondArista es el conjunto finito de etiquetas de condiciones, las cuales se corresponden con las condiciones producidas en los fragmentos usados para describir el protocolo de interacción del elemento software.
- Etiqueta: E $\rightarrow$ EtiqAristas es una función inyectiva de etiquetado, donde EtiqAristas es el conjunto finito de etiquetas que identifican el conjunto de mensajes que pueden recibirse y enviarse, en el elemento software que está siendo descrito.
- Acción: E $\rightarrow$ AcciArista es una función inyectiva que asocia una acción a cada arista del grafo, donde AcciArista es el conjunto finito de etiquetas, que se corresponden con los contadores producidas en los fragmentos usados para describir el protocolo de interacción del elemento software.
- Vert-Inicial $\in$ V es el vértice inicial del grafo.
- Conj_Vert_Finales $\subset$ V el conjunto de vértices finales del grafo.

Definición 2. Sea GI un grafo de interacciones, cada condición cond $\in$ CondArista, debe cumplir la siguiente sintaxis:

- (x # c), (x OP y # c), (x OP y # z OP s), (x # y OP z) donde # denota $\geq$,$\leq$,=, $\neq$,<, o >; OP denota las operaciones de suma y resta; c es un valor numérico; x, y, z y s son variables que toman valores numéricos.
- (x □ c), donde □ denota = o $\neq$; c es un carácter y x es una variable que toma valores de tipo carácter.
- Las condiciones descritas en los apartados anteriores pueden ser compuestas mediante los operadores AND y OR.
- cond $\in$ CondArista, puede presentar valores vacíos si no hay ninguna condición asociada a una arista del grafo.

Definición 3. Sea GI un grafo de interacciones, cada etiqueta etiqa $\in$ EtiqAristas está formado por una tupla <t, n>, donde t describe el tipo del evento (envío ó recepción), n el nombre del mensaje (n $\in$ N, donde N es el conjunto

finito de mensajes del sistema). Una etiqueta etiqa ∈ EtiqAristas, puede presentar valores vacíos si no existe ningún mensaje asociado a una arista. Esta situación se producirá al anidar diferentes fragmentos en un mismo diagrama de secuencias .

Definición 4. Sea GI un grafo de interacciones, cada acción a ∈ Acciones puede tomar la siguiente sintaxis:

- (x = c), donde x es una variable y c es un valor numérico.
- (x = y OP c), donde x e y son variables, OP denota las operaciones de suma y resta y c es un valor numérico.
- a ∈ Acciones puede presentar valores vacíos si no hay ninguna acción asociada a una arista del grafo.

Definición 5. Sea GI un grafo de interacciones, sea v∈ V, entonces cada vértice v es una tupla de la forma <*par*, *ord*, *crit*, *st*, *variables*> donde *par*, *ord* y *crit* son variables de tipo entero positivo usadas para representar paralelismo, secuenciamiento y regiones críticas; *st* es una variable tipo cadena de caracteres, usada para describir el estado en que se encuentra el componente y *variables* es un conjunto de cadenas de caracteres usadas para describir las variables usadas en las condiciones e iteraciones representadas en el grafo.

4. ¿Cómo utilizar los protocolos de interacción?

Una vez obtenidos los protocolos de los participantes, el siguiente paso consiste en explicar cómo podemos utilizarlos. En las plataformas y repositorios de componentes actuales la documentación sobre el uso de un componente se centra en la descripción de su interfaz incluyendo solamente información sintáctica y estructural, obviando información acerca de las restricciones de comportamiento. Nuestra propuesta completa esta información incorporando los protocolos de interacción que permiten verificar su integración en diferentes dominios de uso. No obstante, nuestra aportación no se reduce a ampliar la documentación de los componentes, sino que permite usar esa información para diseñar sistemas que sean más robustos, permitiendo verificar la compatibilidad de los protocolos de interacción de los componentes que los forman o determinar si es posible sustituir un componente por otro en un sistema existente.

Para llevar a cabo verificaciones de este tipo es necesario realizar agrupaciones de componentes. La agrupación permite obtener nuevos protocolos adecuados a un conjunto de componentes, evitando tener que comprobar la adecuación individual de cada uno de ellos. El objetivo es facilitar la reutilización, abstrayendo los detalles internos en los elementos individuales que están siendo agrupados, y centrándose en describir las interacciones con el entorno que les rodea. La agrupación lleva implícita el cumplimiento de las siguientes características:

- Los servicios ofrecidos por la nueva agrupación creada serán la unión de los servicios ofrecidos por los elementos que lo componen.

- Los servicios requeridos por la nueva agrupación, serán la unión de los servicios requeridos por los elementos que lo componen, abstrayendo los servicios que se prestaban entre ellos.

Es posible, aunque técnicamente muy engorroso, definir una operación de agrupación a partir de los grafos que describen los protocolo de interacción. Una vez hecho esto se podrían establecer definiciones de compatibilidad y de reemplazamiento entre grafos. No obstante, para lograr estos propósitos, resulta mucho más sencillo traducir la información de los grafos a descripciones algebraicas. La ventaja de las álgebras de procesos es que su semántica operacional nos permiten definir y comprobar los protocolos de los componentes de una forma más sencilla usando grafos. Resulta trivial obtener una descripción algebraica del protocolo de un componente a partir del grafo correspondiente. Para ello, basta con ir recorriendo dicho grafo: los nodos del mismo se corresponderán con estados en la descripción algebraica, y las etiquetas que figuran en las aristas del mismo determinan los mensajes que se pueden recibir o enviar en un momento dado. Así, para el componente C1 representado en la Figura 1, llegaríamos a la siguiente especificación:

```
C1 = !m1. ?m4. C1sta2
C1sta2 = !m5. ?m8. C1sta3
C1sta3 = 0
```

donde !m representa el envío de un mensaje m (y ?m la correspondiente recepción), el operador punto "." indica secuencia, y 0 representa un estado final (en el que no se envían ni reciben más mensajes).

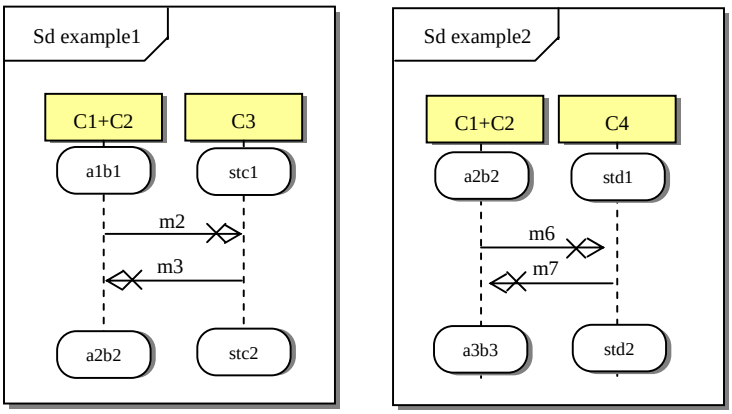

Figura 4. Agrupación de los componentes C1 y C2 de la figura 1.

A partir de estas descripciones algebraicas del protocolo de interacción de cada uno de los componentes del sistema, y de acuerdo con la semántica que define la composición en el álgebra de procesos, es posible obtener el protocolo de interacción correspondiente a la agrupación de varios componentes. Así la figura 4 describe un ejemplo en el cual se representa cómo quedan los diagramas de secuencias de la figura 1, después de agrupar los componentes C1 y C2. Esta agrupación da lugar a un nuevo componente (C1C2 en el ejemplo) que puede ser usado para obtener un protocolo conjunto más sencillo de manejar que los protocolos individuales de cada uno de ellos:

```
C1C2 = !m2. ?m3. C1C2a2b2
C1C2a2b2 = !m6. ?m7. C1C2a3b3
C1C2a3b3 = 0
```

4.1 Sustitución de componentes.

La evolución de los sistemas de información implica que el software que los soporta debe actualizarse continuamente para adaptarse a las nuevas necesidades. En este contexto, la sustitución de componentes software de un sistema es una tarea habitual. Para estudiar la sustitución de un componente de un sistema podemos plantear varias alternativas de trabajo:

1. Podemos analizar si un candidato se comporta exactamente igual que el componente a sustituir. Para ello, se estudia si los protocolos de ambos componentes son equivalentes. En el ámbito de las álgebras de procesos, esto se lleva a cabo mediante análisis de bisimilaridad. Sean P y Q dos protocolos de interacción; se dice que P y Q son bisimilares (y lo representaremos $P \equiv Q$), si y sólo si P es capaz de simular el comportamiento de Q, y Q el de P. Una definición formal de bisimilitud en álgebras de procesos está fuera del ámbito de este trabajo, pudiendo encontrase en la literatura al respecto (véase por ejemplo [MPW92]). Informalmente, podemos decir que para determinar la bisimilitud debemos establecer una relación de equivalencia entre los estados de ambos protocolos de forma que para cada par de estados equivalentes, las

acciones de envío y recepción de mensajes que se pueden realizar en ambos son exactamente las mismas, y en cada caso nos llevan a un nuevo par de estados equivalentes.

2. Obviamente, la definición de equivalencia anterior es demasiado restrictiva, y la mayoría de las veces no será necesario que dos componentes tengan un comportamiento exacto. Para relajar un poco la definición anterior, se puede optar por la noción de similaridad. Sean P y Q dos protocolos de interacción; se dice que P es similar a Q (y lo representaremos $P \subseteq Q$), si sólo si P es capaz de simular el comportamiento de Q, es decir, si para cada par de estados relacionados de P y Q cualquier acción que se pudiese realizar en el protocolo P, sería también posible realizarla en el protocolo Q. De esta forma, aseguraríamos que el procotolo Q incluye al de P, y que por tanto el componente descrito por Q va a poder realizar al menos las mismas acciones que el componente descrito por P (que es el que pretendemos reemplazar). Esta operación además de usarse en la sustitución de componentes, puede ser usada para realizar búsquedas de secuencias parciales de eventos que interese detectar dentro de un protocolo, como pueden ser regiones críticas.

3. No obstante lo anterior, el reemplazo de un componente de un sistema no tiene porque implicar su sustitución por otro con un comportamiento similar. A veces, deseamos hacer un cambio en la funcionalidad del sistema, por lo que un elemento debe ser sustituido por otro cuyo comportamiento es diferente. En estas situaciones, hay que estudiar si el comportamiento del componente es compatible con el entorno donde va a ser integrado. Para realizar este estudio, se puede analizar la compatibilidad individual con cada uno de los componentes con los que se va a relacionar, o estudiar la compatibilidad con todos ellos a la vez. Esta segunda opción, además de ser más rápida y sencilla es la que proponemos en el trabajo. Para realizarla, se agrupan los componentes que se relacionan con el componente sustituido (básicamente, el resto del sistema) y se obtiene su protocolo de interacción. Este se comparará con el protocolo del candidato estudiar su compatibilidad, determinando si pueden derivarse bloqueos de la interacción de ambos componentes. Una definición formal de compatibilidad de protocolos queda fuera del ámbito de este trabajo, pudiendo encontrarse en [CPT01]. Informalmente, podemos caracterizar la compatibilidad del siguiente modo: si P y Q son dos protocolos de interacción, se dice que P es compatible con Q (y lo representaremos $P \rhd\lhd Q$) si no se producen bloqueos como resultado de su interacción, es decir si podemos establecer una correspondencia entre los estados de P y Q de tal manera que en cada par de estados relacionados, si en uno de los estados es posible realizar una acción de salida !a y en el otro es posible realizar la acción correspondiente de entrada ?a, entonces tras realizar ambas acciones llegamos de nuevo a estados relacionados.

4.2 Demostración de propiedades.

Existen situaciones en las cuales es conveniente describir escenarios que expresan propiedades o comportamientos tanto deseados como prohibidos dentro del sistema a desarrollar. En estas circunstancias, se pueden describir diagramas que expresen el patrón de comportamiento a estudiar. Para detectar estos escenarios, no se pueden realizar operaciones de compatibilidad como las estudiadas en los apartados anteriores. Estas propiedades suelen referirse a la interacción de varios componentes que cooperan y cuyos protocolos hay que sincronizar para poder comprobarlas. La sincronización de diferentes protocolos puede ser expresada y detectada utilizando la semántica operacional de las álgebras de procesos con la operación que representa la ejecución en paralelo de diferentes procesos. De esta manera se pueden usar los protocolos de interacción para buscar situaciones complejas, dentro de en un sistema, susceptibles de ser comprobadas.

4.3 Desarrollo de sistemas.

Los diagramas de secuencias permiten profundizar en el conocimiento de los requisitos, al tiempo que permiten describir el comportamiento de los participantes describiendo los mensajes entre ellos. Pero también los diagramas y los protocolos obtenidos se pueden usar para facilitar el desarrollo incremental de sistemas. Para conseguirlo hay que partir de una definición inicial abstracta del sistema e ir incrementando en el nivel de detalle hasta conseguir especificar los componentes requeridos. Este refinamiento progresivo permite llegar a la especificación de

componentes deseados. Estos componentes pueden ser creados u obtenidos de repositorios existentes. En cualquier caso, hay que estudiar como realizar su integración en el sistema. Es decir, hay que realizar operaciones de compatibilidad entre el protocolo de interacciones del componente candidato y el protocolo del entorno donde se integrará. Esta operación se realiza de la misma manera que se ha comentado en el tercer apartado del punto 4.1. Una vez comprobado que no existen bloqueos entre ambos protocolos, se continúa con los demás componentes hasta comprobar la compatibilidad entre todos ellos.

5. Conclusiones y trabajos futuros

Aunque UML es un lenguaje de modelado orientado a objetos, la idea de ampliar su uso para modelar sistemas usando componentes no es ninguna novedad [SW99, CD00]. No obstante, los diagramas de secuencia de UML 1.x eran una herramienta limitada por la falta de fragmentos, corregiones y posibles referencias a otros diagramas. Con la aparición de UML 2.0, los nuevos diagramas de secuencia, basados en los Message Sequence Charts (MSC) propuestos por la International Telecommunication Union [ITUA, ITUB], han ganado en poder de expresividad. Existen propuestas [BK98] en los cuales los MSCs se usan para la descripción de interfaces de interacción, pero se centran en representar el paso de mensajes asíncrono, vía canales directos entre componentes. En este trabajo se plantea la utilización de una semántica síncrona para trabajar con protocolos, sin canales, más sencilla de tratar y más intuitiva que la asíncrona. El uso de una semántica asíncrona requiere la descripción de colas de mensajes asociadas a cada uno de los elementos que intervienen en el sistema, lo cual complica el razonamiento sobre las operaciones entre protocolos. Existen otros trabajos [IU01], que describen el comportamiento de los componentes mediante ecuaciones en CCS, y obtienen grafos de comportamientos a partir de estas, las cuales utilizan para estudiar que los sistemas descritos están libres de bloqueos. No obstante, no utiliza un lenguaje de modelado standard, y se centran sobre todo en la eficiencia de los algoritmos de detección de bloqueos más que en su uso como herramienta de construcción y mantenimiento de sistemas.

Existen otras aproximaciones que sintetizan información de los escenarios, cada una de ellas con diferentes objetivos, como [SDV95] que intenta obtener autómatas de tiempo, o especificaciones en ROOM [LMR98] aunque la mayoría intentan obtener el modelo de estados de los participantes, a partir de la descripción de las interacciones del sistema. Para ello se utilizan diferentes aproximaciones como [WS00] que utiliza variables globales y precondiciones y postcondiciones en los eventos para organizar los escenarios, [UKM03] que utiliza High Message Sequence Charts, [KEK99] que usa diagramas de colaboración y variables de estado [MS01] que aplica una serie de algoritmos que requieren la intervención del usuario. No obstante, estos trabajos modelan el comportamiento y a partir de ahí intentan obtener el modelo de estados. Este trabajo, a diferencia de los anteriores, modela las interacciones del sistema añadiendo en las líneas de vida, de los diagramas de secuencia de cada participante, información sobre el estado en que se encuentra en el momento de realizar la interacción.

Los diagramas de secuencia de UML permiten sintetizar la información de los diagramas en máquinas de estados o descripciones algebraicas. No obstante, la ventaja de la propuesta presentada es que los diagramas de secuencias describen gráficamente las interacciones de los componentes. Esto facilita tanto la descripción, como la comprensión intuitiva de las especificaciones. Los protocolos obtenidos son fácilmente adaptables a las necesidades del desarrollo, de manera que permiten agrupaciones o descomposiciones adecuadas para estudiar la compatibilidad entre las interacciones de los diferentes componentes. Además, de estudiar la compatibilidad en la integración de componentes, el trabajo plantea la idea de usar los protocolos para buscar propiedades dentro de un sistema, detectar escenarios prohibidos y comprobar la similaridad o bisimilaridad entre los comportamientos expresados por dos procesos.

En este artículo no se ha tratado el tratamiento de los fragmentos en los diagramas de secuencias. La inclusión de fragmentos añade nuevas funcionalidades que incrementan la capacidad de descripción, pero también dificultan la síntesis de las trazas obtenidas. Para facilitar esta tarea, existen algunas recomendaciones

- Reducir la utilización de parámetros en los fragmentos, ya que esto implica la utilización de variables dentro de los protocolos, lo que puede complicar las operaciones de compatibilidad posteriores.

- Evitar abusar de los fragmentos anidados, los cuales incrementan la capacidad de descripción a la vez que dificultan su comprensión intuitiva.

- En caso de usar fragmentos, hay que procurar que estos abarquen a todos los participantes del escenario que se está describiendo, sino se pueden producir problemas de interpretación en los protocolos sintetizados.

Entre los trabajos futuros, se pretende la inclusión de restricciones de tiempo dentro del modelo. Aunque existen en UML diagramas específicos para modelar el tiempo, los diagramas de secuencias también pueden ser usados para representarlo. La inclusión del concepto tiempo, implica la modificación de los grafos de interacción así como de las descripciones algebraicas utilizadas para describir las secuencias de eventos.

6. Referencias

[B78] G. V. Bochmann. “Finite State Description of Communication protocols”. Proceedings of Computer Network Protocols Symposium. Bélgica Feb 1978.

[BHS91] F. Belina, D. Hogrefe, A. Sarma. “SDL with applications from protocols specification”. Ed. Prentice Hall,1991. ISBN: 0-13-785890-6.

[BK98] M. Broy, I. Krüger. “Interaction Interfaces Towards a scientific foundation of a methodological usage of Message Sequence Charts”. In J. Staples, M. G. Hinchey and Shaoying Liu, editors, Formal Engineering Methods (ICFEM´98), pages 2-15. IEEE Computer Society.

[BT82] G. Berthelot and R. Terrat . “Petri Nets Theory for the Correctness of Protocols”. Proceedings of Protocol Specification, Testing and Verification II, pp 325-341, Los Angeles, North-Holland (1982).

[BZ83] D. Brand, P. Zafiropulo “On Communicating Finite-State Machines”. Journal of ACM, vol 30, nº 2. pags. 323-342. Abril 1983.

[CD00] J. Cheesman, J. Daniels. “UML Components, A Simple Process for Specifying Component-Based Software”. Addison-Wesley, Pearson Education October 2000. ISBN 0-201-70851-5.

[C01] C. Canal, E. Pimentel, J.M. Troya, A. Vallecillo. “Extending Corba Interface with Protocols”. The Computer Journal, Vol. 44, Nº 5, pags. 448-462. 2001.

[CPT01] C. Canal, E. Pimentel, J.M. Troya,. “Compatibility and Inheritance in Software Architectures”, Science of Computer Programming, 41(2):105–138, Octubre 2001.

[D82] M. Diaz, “Modelling and Analysis of Communication and Cooperation Protocols Using Petri Net Based Models”. Proceedings of Protocol Specification, Testing and Verification II, pp 419–441, Los Angeles, North-Holland (1982).

[GAO95] D. Garlan, R. Allen, J. Ockerbloom. “Architectural Mismatch or Why it´s hard to build systems out existing parts”. Proceeding 17th International Conference on Software Engineering. Abril de 1995.

[GB98] C. Gacek, B. Boehm. “Composing Components: How Does One Detect Potencial Architectural Mismatches?”. Position paper to the OMG-Darpa-MCC Workshop on Compositional Software. January, 1998.

[IU01] P. Inverardi, S.Uchitel. “Proving Deadlock Freedom in Component-Based Programming” Proceedings FASE 2001, LNCS 2029 , Genova April 2001

[ITUA] ITU-TS. Recommendation Z.120:Message Sequence Chart (MSC). Geneva 1996.

[ITUB] ITU-TS. Recommendation Z.120 : Annex B. Geneva 1998.

[J00] J. Han (2000). “Temporal logic based specifications of component interaction protocols”. In Vallecillo, A., Hernández, J., and Troya, J. M., editors, Proc. of the ECOOP’2000 Workshop on Object Interoperability (WOI’00), pages 43–52.

[MMM94] R. Mili, R. Mittermeir, A.Mili. “Storing and Retrieving Software Component: A Refinement Based Approach”. Proceeding 1th Internacional Conference Software Engineering. 1994.

[MPW92] Milner, R., Parrow, J., y Walker, D. “A Calculus Processes”, Journal of Information and Computation, 100:1–77, 1992.

[PB96] Y. Pai and P. Bai. “Retrieving software components by execution”. Proceedings of the 1st Component User Conference, Munich 1996. Pag 39-48.

[PP93] A. Podgursky, L. Pierce. “Retrieving reusable software by sampling behaviour”. ACM Transactions on Software Engineering and Methodology 2(3),286-303, 1993

[S96] P. Steyaert, C. Lucas, K. Mens, T. D´Hondt. “Reuse Contract: Managing the evolution of reusable assets.” ACM Sigplan Notices, Vol. 31(10), pags 268-285. ACM Press 1996.

[SW99] D. Francis D´Souza, A. Cameron Wills. “Objects, Components and Frameworks with UML: The Catalyisis Approach”. Objects Technology Series. Ed. Addison-Wesley, 1999. ISBN: 0-201-31012-0.

[T93] K. J. Turner. “Using formal description techniques: an introduction to Stelle, Lotos and SDL”. Ed. John Wesley and Sons, 1993. ISBN: 0-471-93455-0.

[UML] Sitio web de UML. http://www.uml.org.

[YS97] D. M. Yellin, R. E. Strom. “Protocol Specification and Component Adaptors”. ACM Transaction on Programming Languages and Systems, Vol. 19, Nº 2, Marzo 97, pag. 292-333.

[ZW95] A. M. Zaremski, J. M. Wing. “Signature Matching, a Tool for Using Software Libraries”. 3rd ACM SIGSOFT Symposium on the Foundations of Software Engineering, pages 6-17. October 1995. Also CMU-CS-95-127.

[SDV95] Somé, S., Dssouli, R. and Vaucher, J., From Scenarios to Timed Automata: Building Specifications from User Requirements in Asia Pacific Software Engineering Conference (APSEC'95), (1995), 48-57.

[LMR98] Leue, S., Mehrmann, L. and Rezai, M., Synthesizing ROOM Models from Message Sequence Charts Specifications in 13th IEEE Conference on Automated Software Engineering (ASE'98), (Honolulu, 1998), IEEE CS, 192-195.

[WS00] J. Whittle and J. Schumann. Generating Statechart Designs From Scenarios. Proceedings of OOPSLA 2000 Workshop: Scenario based round-trip engineering, October 2000.

[UKM03] Uchitel, S., Kramer, J. and Magee, J. Synthesis of Behavioural Models from Scenarios. IEEE Transactions on Software Engineering, 29 (2). 99-115. 2003.

[KEK99] Khriss, I., Elkoutbi, M. and Keller, R., Automating the Synthesis of UML StateChart Diagrams from Multiple Collaboration Diagrams in UML'98: Beyond the Notation, (1999), Springer-Verlag, 132-147.

[MS01] Mäkinen, E. and Systä, T., MAS – An Interactive Synthesizer to Support Behavioral Modeling in UML, in 23rd IEEE International Conference on Software Engineering (ICSE '01), (Toronto, 2001), 15-24.