Síntesis de patrones de interacción a partir de diagramas de secuencia en UML.

Miguel A. Pérez, Amparo Navasa, Juan M. Murillo
Universidad de Extremadura, Departamento de Informática, Grupo Quercus de Ingeniería del Software,
{toledano, amparonm, juanmamu}@unex.es
Carlos Canal
Universidad de Málaga, Departamento de Lenguajes y Ciencias de la Computación, Grupo GISUM,
canal@lcc.uma.es

Resumen

La reutilización de componentes es en la actualidad una de las principales línea de trabajo dentro de la Ingeniería Software. Su utilización en la construcción de sistemas requiere la búsqueda y selección de componentes candidatos, adecuados a los requisitos planteados. Una vez obtenido un candidato, es necesario estudiar la compatibilidad entre las interacciones de este y las del sistema donde va a ser integrado. Este trabajo plantea el uso de los Diagramas de Secuencia, incluidos dentro del lenguaje de modelado UML, como método para obtener los patrones 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 patrones para componentes individuales o para agrupaciones. Los patrones obtenidos, descritos mediante máquinas de estados extendidas y descripciones algebraicas, completarán la información de los interfaces de cada uno de los componentes del sistema. Esta información se podrá usar posteriormente para estudiar la compatibilidad del componente con el entorno donde va a ser integrado, detectar secuencias de eventos prohibidas y realizar operaciones de simulación.

1. Introducción.

Los rápidos cambios producidos en los sistemas de información de las organizaciones han disminuido el tiempo y los recursos que son posibles asignar 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 hipoteque la calidad de las soluciones creadas. En este contexto la reutilización, y más concretamente el desarrollo de software basado en componentes, se ha perfilado como línea de trabajo prioritaria.

La composición de sistemas, usando componentes, implica la búsqueda y selección de aquellos que proporcionen los servicios requeridos dentro del sistema. Además, hay que establecer una verificación sintáctica que permita comprobar si la sintaxis, mediante la que los servicios son proporcionados, coincide con la utilizada por los requisitos. Pero a pesar de lo anterior, siguen pudiendo surgir problemas de integración [1] 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 entorno donde va a ser integrado. Es decir, comprobar si los patrones de interacción de ambos son compatibles.

El patrón de interacción de un componente software es la secuencia ordenada de eventos y restricciones que describen su comportamiento correcto. Existen diferentes enfoques para modelar y verificar estos patrones, que van desde el uso de lenguajes formales específicos [2] hasta diversos tipos de formalismos, tales como máquinas de estados [3], redes de Petri [4], álgebras de procesos [5], lógica temporal [6], contratos de reuso [7] y ontologías [8]. La representación de las interacciones entre componentes usando estas técnicas no permite la comprensión intuitiva de las especificaciones obtenidas, ni están integradas como una tarea más dentro del modelado del sistema. Además muchas de ellas requieren al desarrollador el conocimiento de herramientas de especificación complejas.

Este trabajo persigue integrar la obtención de los patrones como una tarea más del diseño del

sistema usando UML, sin que se tengan que realizar descripciones adicionales al modelado. Para ello se usarán los diagramas de secuencia, ya que es 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. De esta forma se completa la información de los interfaces [9], obteniendo patrones de interacción adecuados a cada uno de los componentes del sistema. Estos patrones de interacción serán usados para realizar operaciones de verificación durante la construcción y mantenimiento de sistemas basados en componentes. Además, la propuesta permite la agrupación de componentes para obtener patrones de interacción adecuados a grupos. Los patrones de interacción de estas agrupaciones contendrán las interacciones de la agrupación con el entorno, eliminando las referencias internas entre los componentes agrupados. Esto facilitará las operaciones de integración así como el incremento de la granularidad de los elementos reutilizados.

Para obtener los patrones, se describen todos los diagramas de secuencia necesarios para representar las interacciones entre los componentes que intervienen 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 utilidades diferentes. Por una parte, la construcción de máquinas de estados extendidas permite detectar inconsistencias en las especificaciones, como se verá en el apartado 3, y pueden ser usados para operaciones de simulación, ya que permiten representar toda la metainformación aportada por los diagramas (invariantes, regiones críticas, variables...). No obstante, para facilitar las operaciones con patrones, los grafos pueden ser complementados usando descripciones algebraicas que permiten describir, de una manera más sencilla, la colaboración entre varios componentes al mismo tiempo.

La estructura de este trabajo es la siguiente: en la Sección 2 se explica brevemente como utilizar los diagramas de secuencias para poder, posteriormente, sintetizar de manera correcta las especificaciones descritas; en la Sección 3 se explican los pasos necesarios para la creación de los patrones de interacción a partir de las especificaciones de los diagramas de secuencias; la Sección 4 presenta cómo usar estos patrones para el análisis de diversas propiedades; en la Sección 5 se exponen las conclusiones y trabajos relacionados y por último se presenta la bibliografía utilizada.

2. Consideraciones en la construcción de los diagramas de secuencias.

Los diagramas de secuencias son una herramienta gráfica fácil de usar, que genera especificaciones sencillas de comprender. No obstante, presentan algunas limitaciones que hay que intentar matizar para poder usarlos correctamente como herramienta que permita la obtención de patrones de interacción, y que podemos resumir en [10]: ofrecen una visión parcial del sistema, suelen describir interacciones entre participantes a nivel de instancia, y además explicar todas las interacciones puede provocar una explosión en el número de escenarios.

Como hemos comentado, cada escenario proporciona una visión parcial que hay contextualizar con los demás escenarios, para de esta manera poder tener una idea global del sistema. Para resolver este problema se pueden utilizar los diagramas de visión general (Overview Diagrams) de UML, que 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 no de todo el sistema como hacen los Overview Diagramas. Para poder estudiar la evolución individual de cada participante, se propone el uso de las etiquetas de invariantes de UML como indicadores del estado en el que se encuentra el componente. Esto hará que dentro del diagrama de secuencias se modelen las interacciones junto con los estados. Para ello, la línea de vida (proyección de las interacciones de un participante dentro de un diagrama) de cada participante debe comenzar y terminar con una etiqueta que indique el estado en el que comienza y termina el componente dentro del escenario descrito. Por lo tanto, las etiquetas tienen dos cometidos: describir la evolución de los estados

por los que pasa un componente, y servir de nexo de unión entre las especificaciones dispersas en diferentes diagramas.
Otra de las críticas habituales sobre los escenarios, es que generalmente describen interacciones entre ocurrencias de los diferentes tipos de participantes. Como los patrones de interacción se obtienen para cada componente del sistema y no para cada ocurrencia, hay que organizar la información agrupándola por tipos genéricos de componentes, abstrayendo las ocurrencias individuales. Por último, el problema de la explosión en el número de escenarios puede ser controlado modelando los estados junto a las interacciones, y usando las nuevas funcionalidades de UML 2.0 como la reutilización de escenarios y los fragmentos.

3. Creación de patrones de interacción.

Los siguientes apartados irán enfocados a describir como obtener patrones de interacción adecuados a componentes o conjuntos de componentes. Las tareas a realizar son las siguientes: la construcción de los diagramas, el testeo de las especificaciones descritas y la síntesis de los patrones de interacción.

3.1. Construcción de diagramas.

Para la creación de patrones de interacción, la primera tarea a realizar consiste en modelar el sistema utilizando las herramientas proporcionadas por UML. Así, una vez realizada la descripción de requisitos, utilizando diagramas de casos de uso, es necesario diseñar correctamente el conjunto de diagramas de secuencias que describen su funcionamiento. En este punto es importante identificar todos los escenarios necesarios para describir cada uno de los requisitos e identificar los componentes que intervienen en ellos. En este sentido, igual que en [11], las situaciones a describir son los escenarios que ilustren los comportamientos que obligatoriamente u opcionalmente debe cumplir el sistema, y adicionalmente se pueden describir escenarios que describan comportamientos prohibidos.
Para comprender mejor la propuesta desarrollaremos un ejemplo práctico que explique los pasos a dar. Se pretende describir el funcionamiento de una máquina expendedora de bebidas (Figura 1). Para simplificar el ejemplo, el funcionamiento se ha restringido a dos situaciones: “vender bebida” y “mantenimiento de la máquina” en los cuales se han identificado tres componentes iniciales: el panel de entrada de peticiones, el depósito donde están guardadas las bebidas y el monedero que contiene el dinero recaudado. En la primera situación, “vender bebidas”, una vez recibida una petición, se comprueba que tenemos disponible la bebida solicitada, se realiza el cobro (suponemos que se introduce el precio exacto) y se entrega la bebida. La segunda situación, “mantenimiento de la máquina”, añade bebidas al depósito y recauda el dinero. Por problemas de espacio, se ha representado que la venta se realiza sin ninguna excepción, aunque se debería completar la especificación con otros escenarios que describiesen otras situaciones posibles, como que no exista la bebida o que el cobro no se pueda realizar correctamente. En cuanto al modelado de los estados de los componentes que intervienen, el componente panel se encuentra inicialmente en el estado de “espera” y sólo sale de él cuando recibe un mensaje para reponer la máquina y pasa al estado de “bloqueo”; el componente depósito comienza inicialmente “vacío”, cuando recibe las bebidas pasa al estado de “listo”, al servir una bebida al estado de “servicio” y cuando se vacía vuelve al inicio. Por último, el componente monedero inicialmente está en le estado de “espera”, y solo evoluciona cuando cobra una bebida y pasa al estado de “cobro”.

3.2. Testear especificaciones.

Una vez descritos todos los diagramas de secuencias hay que estudiar las especificaciones para cada componente que interviene en el sistema, ya que pueden presentar redundancias e inconsistencias. Esto implica seleccionar los escenarios en los que participan y obtener el comportamiento expresado en cada uno de ellos. Por lo tanto existirán para un mismo componente toda una colección de trazas que hay que coordinar usando las etiquetas.

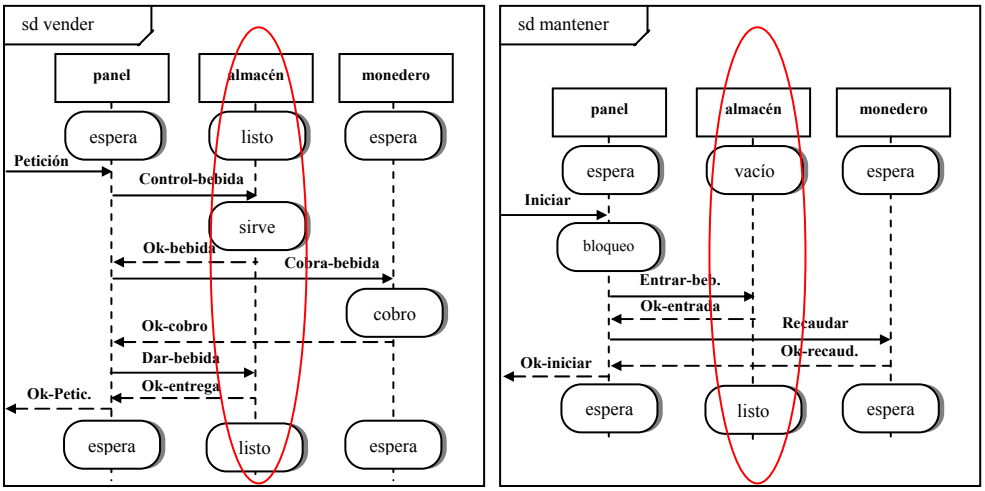

Figura 1. Diagramas de secuencia usados para describir el funcionamiento de la máquina de bebidas

Todos los componentes evolucionan desde un estado inicial, y van hilvanando una secuencia de eventos que van determinando su comportamiento. En estas condiciones, hay que ordenar 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 hay que seleccionar la que represente el estado inicial del participante y a partir de ahí se irán concatenando la ejecución de las demás trazas. Esta tarea se divide en varios pasos:

- Comprobar la completitud de las especificaciones. Una vez obtenidas todas las trazas de un componente, es probable que estas no se puedan unir. Esto puede ser debido a un problema en la especificación, causado por la falta de escenarios o porque existe un conflicto en la descripción de las etiquetas de alguna de las trazas que habrá que corregir.
- Otro de los chequeos a realizar consiste en estudiar la coherencia de las especificaciones descritas. Para ello, aprovechando que los diagramas de secuencias están incluidos en una herramienta de modelado como UML, se contrastan las especificaciones obtenidas en los diagramas de secuencias con las obtenidas en otros diagramas de UML, sobre todo diagramas de estados, de componentes y de clases. Este estudio permite detectar errores en la especificación de los estados representados en las etiquetas, inconsistencias, carencia de algún escenario, errores en los mensajes, e incluso errores en la secuencia de eventos descritos.
- El último paso consiste en eliminar redundancias. Aunque se ha intentado minimizar usando fragmentos y reutilizando escenarios. En la descripción de escenarios es frecuente que se repitan secuencias de eventos. No obstante, las redundancias no constituyen ningún problema, ya que pueden ser fácilmente detectadas y eliminadas en la construcción de los grafos obtenidos durante la síntesis de los diagramas de secuencias, que se describe en el apartado siguiente.

Como excepción, hay que comentar que los escenarios que expresan secuencias de eventos prohibidas, usando el fragmento “negative”, deben marcarse y tratarse de manera independiente cada uno de ellos. Estos escenarios no se usan para crear patrones de interacción, sino para validar si los sistemas descritos cumplen las restricciones definidas dentro de ellos. Además, a estos escenarios no se le aplicarán las mismas operaciones de testeo comentadas anteriormente, ya que no existe la posibilidad de realizar las mismas comprobaciones que las realizadas con las especificaciones ordinarias.

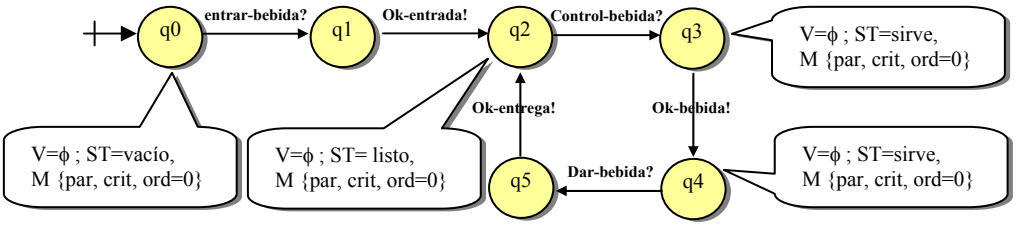

Figura 2. Grafo resultado de sintetizar el componente "almacén"

3.3. Síntesis de los patrones de interacción.

La síntesis completa de la información descrita en los diagramas de secuencias (incluyendo secuencias, fragmentos, corregiones, parámetros y estados) permite obtener una máquina de estados extendida, concretamente un grafo orientado etiquetado y cíclico. Para comprender mejor la creación de grafos se presenta el grafo resultante de sintetizar las especificaciones del componente almacén (figura 2). En él todas las aristas son del tipo evento, precedidas del carácter "?" para indicar recepción, y del carácter "!" para indicar envío. Además, para simplificar sólo se ha incluido información de 4 vértices siendo los dos que faltan similares.

Los grafos están formados de vértices que contienen información sobre las variables que se usan dentro de los diagramas de secuencias (conjunto "V"), el estado del componente en el que se encuentra la transición (variable "ST"), así como una serie de variables internas que toman valores enteros positivos y describen la ordenación ("ord"), el paralelismo ("par") y las regiones críticas ("crit"). Así mismo, cada arista puede contener tres tipos de informaciones distintas: condiciones, eventos y acciones. Las condiciones se usan para poder representar las diferentes opciones de los fragmentos de los diagramas de secuencias ("alt", "opt", "break", "loop"), los eventos representan los mensajes de los diagramas y por último las acciones sirven para inicializar e incrementar los contadores necesarios para describir las iteraciones del fragmento denominado "loop". Una mayor información sobre la utilización de los grafos se puede consultar en [12].

Los grafos sintetizan completamente la información de las trazas de los componentes permitiendo testear las especificaciones, así como realizar simulaciones del sistema descrito, no obstante presentan algunas limitaciones. En primer lugar, el volumen de información y la estructura de datos obtenida dificultan las operaciones entre grafos. Y en segundo lugar, los grafos representan el patrón de interacción de un componente dentro de un sistema software, sin embargo no pueden ser usados para describir escenarios. Es decir, cuando se describe un escenario con una secuencia de eventos prohibida, no es correcto obtener un grafo de cada componente que interviene en él, ya que son secuencias incompletas y fuera de contexto, y hay que sintetizar el escenario globalmente.

Para salvar estas carencias, la síntesis de las interacciones se puede realizar además utilizando descripciones algebraicas que aligeran la información de los grafos y permiten la descripción de componentes individuales o escenarios completos. Además, existen herramientas en el mercado que permiten validar y comparar especificaciones escritas en álgebras de procesos como CCS [13]. De esta manera, los grafos y las descripciones algebraicas se complementan para poder realizar todas las operaciones de simulación, validación, verificación y testeo necesarias dentro del sistema descrito.

Siguiendo con el ejemplo, el grafo descrito en la figura 2, podría ser representado usando la representación en CCS como:

```
vacío=Add-drink?. Ok-add.ready! | 0
listo=Check-drink?. service!
sirve=Ok-drink!. Give-drink?. Ok-give!.ready
```

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

Además, para facilitar las operaciones de compatibilidad entre los patrones de interacción de los componentes de un sistema software, es conveniente permitir diferentes niveles de abstracción. Esto puede realizarse agrupando componentes de manera que se puedan obtener nuevos patrones adecuados a estas agrupaciones. El objetivo es abstraer los detalles internos de los elementos individuales que son agrupados, y centrarse 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.

4. Utilización de los patrones de interacción.

La creación y evolución de sistemas en desarrollos basados en componentes, implica que las operaciones de integración y sustitución de componentes sea una tarea habitual. Para estudiar estos problemas los patrones de interacción pueden ser tratados de diferentes maneras:

- Podemos analizar si un candidato se comporta exactamente igual que el componente a sustituir. Para ello, se estudia si los patrones de interacción 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 patrones 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 a Q, y Q a P. Entenderemos por simular, al hecho de que un patrón sea capaz de realizar las mismas acciones que otro. Una definición formal de bisimilitud en álgebras de procesos está fuera del ámbito de este artículo, pudiendo encontrase en la literatura al respecto (véase por ejemplo [14]).
- No obstante, la definición de equivalencia anterior es bastante 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 patrones 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 patrón P, sería también posible realizarla en el patrón Q. De esta forma, aseguraríamos que el patrón 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 patrón, como pueden ser regiones críticas.
- 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 nuevo 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, es más rápida y sencilla. Para realizarla, hay que agrupar los componentes que se relacionan con el componente sustituido (básicamente, el resto del sistema) y se obtiene su patrón de interacción. Este se comparará con el patrón del candidato para estudiar su compatibilidad, determinando si pueden derivarse bloqueos de la interacción de ambos componentes. Una definición formal de compatibilidad de patrones queda fuera del ámbito de este artículo, pudiendo encontrarse en [9]. Informalmente, podemos caracterizar la compatibilidad del siguiente modo: si P y Q son dos patrones de interacción, se dice que P es

compatible con Q (y lo representaremos P $\triangleright\triangleleft$ 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.

5. Trabajos relacionados y conclusiones.

En este trabajo se ha presentado como sintetizar los diagramas de secuencias de UML para obtener patrones de interacción adecuados a cada uno de los componentes de un sistema software. Para realizar esta síntesis de manera completa se ha planteado el uso de grafos etiquetados cíclicos y dirigidos, los cuales pueden ser importados y simulados en la herramienta Active-HDL v6.3 [15], así como de descripciones algebraicas cuyas operaciones pueden realizarse en la herramienta Concurrency Workbench of the New Century [13]. El uso de ambas tiene como objetivo facilitar las operaciones de integración o sustitución de componentes software del sistema.

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 patrones de interacción que permiten verificar su integración en diferentes dominios de uso. No obstante, esta 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 patrones de interacción de los componentes que los forman o determinar si es posible sustituir un componente por otro en un sistema existente.

Para construir los patrones de interacción, usando diagramas de secuencias, se ha propuesto integrar el modelado de las interacciones con el de los estados de los componentes del sistema. Para ello se han utilizado etiquetas en la línea de vida de los participantes como ya habían hecho, con diferentes objetivos a trabajos anteriores [16,17]. De esta forma, se han reducido los huecos entre los diagramas de secuencias y de estado en UML, al tiempo que se ha relacionado la ejecución dispersa en diferentes escenarios.

Existen otras aproximaciones que sintetizan información de los escenarios, cada una de ellas con diferentes objetivos, como [18] que intenta obtener autómatas de tiempo, o especificaciones en ROOM [19] 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 [20] que utiliza variables globales y precondiciones y postcondiciones en los eventos para organizar los escenarios, [21] que utiliza High Message Sequence Charts, [22] que usa diagramas de colaboración y variables de estado y [23] 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 y los estados en los diagramas de secuencia. De esta manera, se documentan las interacciones y se describe cual es el estado en el que se encuentra cada participante en el momento de realizarlas.

En cuanto a los problemas de descripciones a nivel de instancias y elevado número de escenarios necesarios para describir el sistema, que presenta la síntesis de escenarios [24], nuestro trabajo propone la generalización de la información de los diagramas de secuencias para tener descripciones de tipos diferentes de componentes. Por otra parte, la agrupación de componentes permite incrementar la granularidad y disminuir el número de escenarios, elevando o disminuyendo el nivel de abstracción según las necesidades del momento.

Entre los trabajos futuros se pretende complementar la propuesta con la inclusión de restricciones de tiempo dentro del modelo. Para ello, se trabaja con contadores de tiempo que completen la información de los grafos etiquetados cíclicos dirigidos, y permitan representar las restricciones de tiempo descritas en los diagramas de secuencia.

Referencias

[1] 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.

[2] 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.

[3] W. Peng: Deadlock Detection in Communicating Finite State Machines by Even Reachability Analysis. Special Issues: Protocols for Mobile Environment. Kluwer Academic Publisher. ISSN: 1383-469. Pages 251-257.

[4] R.S. Cost, Y. Chen, T. Finim, Y. Labrov, Y. Peng: Using Coloured Petri Nets for Conversation Modeling. Issues in Agent Communication. Springer-Verlag, Heidelberg, Germany. Pages 178-192.

[5] 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.

[6] J. Han: Temporal logic based specifications of component interaction protocols. Proc. of the ECOOP'2000 Workshop on Object Interoperability (WOI'00), pages 43–52.

[7] 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.

[8] S. Cranefield, M. Purvis, M. Nowostawski, P. Hwang: Ontologies for Interaction Protocols. In Proceeding of the Workshop on Ontologies in Agent. 2002.

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

[10] H. Behrens: Requirements Analysis and Prototyping using Scenarios and Statecharts. In Proceedings of ICSE 2002 Workshop: Scenarios and State Machines: Models, Algorithms, and Tools. 2002.

[11] Ø. Haugen and K. Stølen: STAIRS - Steps To Analyze Interactions with Refinement Semantics. In UML 2003. San Francisco, Springer-Verlag, LNCS 2863 pp 388-402.

[12] M.A. Pérez, A. Navasa, J.M. Murillo. "Conversión de la Información obtenida a partir de diagramas de secuencia de UML en grafos de comportamiento". Technical Report: TR-22/2004. Universidad de Extremadura.

[13] Concurrency Workbench of the New Century (CWB-NC). http://www.cs.sunysb.edu/~cwb

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

[15] Active-HDL v6.3. Aldec FPGA verification. http://www.aldec.com/ActiveHDL

[16] Ø. Haugen: Using MSC-92 effectively. In Proceeding 7º SDL Forum, pages 37-49, 1995, Amsterdam.

[17] F. Meijs: Message Sequence Chart Enhancements. Technical Report RWB-506-ir-95071, Philips research, 1996.

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

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

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

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

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

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

[24] S. Uchitel, R. Chatley, J. Kramer, J. Magee: System Architecture: the Context for ScenarioBased Model Synthesis. ACM International Symposium on Foundations of Software Engineering (FSE'04), 2004.