Especificación de Patrones de Diseño de Event-B como procesos NTCC

C. E. Gómez Hernández[1]

[1] Departamento de electrónica y ciencias de la computación, Pontificia Universidad Javeriana Cali, Colombia. cegomez@javerianacali.edu.co

Resumen—Los métodos formales son importantes para modelar sistemas críticos, ya que con ellos se tiene una certeza del correcto funcionamiento de dichos sistemas. Por ejemplo, en el contexto de cirugías asistidas por computador, uno esperaría que cada acción del robot corresponda a la acción que el usuario (en este caso el doctor) esté realizando, después de todo, está en juego la vida de una persona.

Dentro del diseño de software, Event-B usa modelos llamados patrones de diseño, y tal como se realiza en programación clásica, el reutilizar código es importante para hacer mejores programas y evitar pensar en problemas que previamente fueron resueltos. Event-B usa los patrones de diseño para reutilizar modelos que previamente han sido verificados, haciendo el proceso de modelamiento más eficiente.

El proyecto de grado entonces explorará estos modelos, analizará sus semejanzas y propondrá un modelo de traducción entre modelos. Para realizarlo propondrá un lenguaje de alto nivel para que el diseñador le sea más sencillo especificar dichos patrones. Este lenguaje, como veremos, se puede traducir de manera simultánea a los dos tipos de especificaciones mencionados anteriormente manteniendo la estructura general del modelo que se quiere especificar. Así, la tarea del modelador resulta más sencilla utilizando el método propuesto en este trabajo de grado.

I. INTRODUCCIÓN

En el mundo de los métodos formales, existe un sin número de lenguajes que permiten modelar los sistemas de la vida real basado en lenguajes matemáticos precisos. Entre estos lenguajes, se destacan dos que son el objeto de interés de este proyecto: el lenguaje de procesos concurrentes NTCC [1] y el framework de modelado de sistemas Event-B [2].

El lenguaje NTCC (Non-Deterministic Concurrent Constraint Programming) es un lenguaje que permite modelar procesos concurrentes de manera no determinista, así como hacer pruebas sobre el modelo creado, todo esto basado en la teoría de cálculos de procesos.

Al diseñar un sistema, hay que tener en cuenta el tiempo en que cada proceso debe ser ejecutado. El diseño de patrones de diseño requiere planear de cierta manera la secuencia en que cada proceso debe ser ejecutado, y esta planeación a la larga se hace de manera “manual". Esta forma de realizarlo genera un desgaste en tiempo de implementación ya que estos patrones pueden ser reutilizados para ciertos modelos en general.

Por esta razón, el presente trabajo va encaminado a buscar una forma que permita reutilizar patrones de diseño, empezando desde un modelo en NTCC que pueda ser luego esquematizado (modelado como una estructura general) en Event-B. Para realizar tal tarea, el conocimiento de éstos lenguajes formales es indispensable, así como el conocimiento de los patrones de diseño ya existentes ya que con ellos, se desea buscar ciertos patrones capaces de ser utilizados para muchos problemas.

Para esto vamos a utilizar un lenguaje de alto nivel que permite dar al diseñador una facilidad a la hora de modelar estos patrones gracias a su estructura que permite definir de manera ``natural" dichos modelos. Además de lo anterior, el lenguaje de alto nivel permite

generalizar los modelos tanto en NTCC como en Event-B.

II. FUNDAMENTACIÓN TEÓRICA

A. Marco Teórico.

- Patrones de Diseño.

Los patrones de diseño (de ahora en adelante PD), son un concepto de ingeniería cuyo objetivo es facilitar el desarrollo de sistemas. Tal como lo dice [3], los PD no necesariamente son un producto terminado, pero son un *template* de cómo resolver problemas que se pueden usar en diferentes situaciones. Así que se puede decir que los PD son una estructura general de ciertas propiedades de un sistema. Un patrón de diseño clásico es el MVC o Modelo Vista Control, donde el Modelo es la representación de la información que posee el sistema como los accesos a ella, la vista es la parte gráfica del modelo (Interfaz de Usuario) y el controlador responde a las acciones que le manda la vista.

- CCP.

Concurrent constraint programming, es un formalismo de concurrencia basado en el modelo de compartimiento de variables. En [1], se menciona que sus orígenes están dados por las ideas de ``computar con restricciones", ``programación lógica concurrente" y ``programación lógica por restricciones". Lo anterior es la base en la cual se ha creado CCP que permite realizar pruebas matemáticas sobre el modelo para verificar su correctitud.
CCP ha venido siendo desarrollado durante más de 20 años. En [4] explican su estructura y funcionamiento y en [1] mencionan el estado de arte completo de este formalismo. CCP está basado en el cálculo de procesos. Éste lenguaje utiliza un pequeño conjunto de operadores primitivos que le permite realizar un sin número de operaciones y modelos concurrentes (lo que lo hace atractivo a los modeladores). En este lenguaje se tiene un Store que es el lugar donde todos los *contraints* se encuentran y donde los procesos pueden realizar sus operaciones con los operadores del modelo. Entre estos operadores se encuentran el ``Operador Tell" y el ``Operador Ask" que guarda y pregunta respectivamente un constraint en el Store. También existe la ``Composición Paralela", que simplemente combina procesos concurrentes (realiza más de 1 proceso) y un operador de localidad que introduce variables locales y restringe las interfaces de los procesos que interactúan con los demás, es decir, solo actúa con ciertos procesos.
Con todo lo anterior CCP puede entonces modelar procesos reactivos de manera formal y hacer verificaciones sobre el modelo para evitar estados o comportamientos no deseados sobre este.

- TCC.

TCC, o *Temporal Concurrent Constraint Programming* es un formalismo que combina la idea de CCP con los lenguajes sincrónicos [1].
En [1] mencionan que TCC está dividido en *intervalos discretos*, esto es, unidades de tiempo discretas, que por cada tiempo un proceso determinístico CCP es ejecutado (recibe un estímulo) y aquel proceso que no pueda ser ejecutado, se espera hasta la otra unidad de tiempo para ejecutarlo si está bajo un operador Next.
TCC extiende del modelo de CCP con los operadores temporales Next P y unless c Next P. El primer operador indica que el proceso P se ejecutará en la próxima unidad de tiempo y el segundo operador indica que el proceso P se ejecutará en la próxima unidad de tiempo al menos que *c* pueda deducirse del Store al final del intervalo de tiempo presente.

- NTCC.

NTCC proviene de las palabras *Non-Deterministic Time Concurrent Constraint Programming* [1]. NTCC extiende los operadores de TCC que a su vez extiende a CCP [1].
NTCC es una extensión del modelo de TCC. Esto implica que NTCC obtiene todos los operadores de TCC (que incluyen los de CCP) y además extiende a TCC agregando *guarded-choices* que son guardas para que se seleccione de manera no determinista los procesos [5].

- Event-B

Event-B es un método formal de modelado de sistemas [2]. Sus principales características están dadas por su uso de la teoría de conjuntos para hacer el modelo y las pruebas matemáticas. Usa refinamientos como método de abstracción del modelo y las pruebas matemáticas garantizan la consistencia entre los distintos niveles de atracción.
Event-B fue creado por Jean-Raymond Abrial [2], un científico de la computación francés conocido por ser el inventor de los métodos formales Z y B. Luego de concebir B, creó Event-B, para realizar operaciones de B con una interfaz amigable.
Los modelos creados por Event-B son descritos en términos de dos constructores básicos: contextos y máquinas [2]. Los contextos contienen la parte estática del modelo, mientras que las máquinas contienen la parte dinámica. Los contextos pueden tener los conjuntos, constantes, axiomas, donde los conjuntos pueden ser similares a tipos. Las máquinas en cambio, contienen variables, invariantes y eventos.

B. Trabajos relacionados.

En [3], Hoang et al. describen la forma de modelar patrones de diseño en Event-B. Al inicio da una introducción al modelamiento en Event-B, describe formalmente los problemas que va a tratar y describe la forma de como los desarrolla en Event-B.
Este paper es importante para el desarrollo del proyecto ya que, además de dar un resumen al modelamiento en Event-B, describe y explica un sistema que incorpora patrones de diseño, además muestra la manera de usar este patrón de diseño en otro sistema con ayuda de una herramienta propia de Event-B. Tambien, este paper está muy ligado al presente trabajo porque tiene el mismo objetivo: modelar patrones de diseño en Event-B, solo que el paper no menciona en ninguna parte el modelo de PD en NTCC, que es la extensión que haría el presente proyecto.
Sin embargo, a diferencia de [3], la idea es utilizar un lenguaje de alto nivel para poder modelar estos patrones de diseño. Este lenguaje proporcionará una forma más natural de representar el modelo de patrones de diseño y, bajo el mismo objetivo de lo que se expresa en este paper, el lenguaje permitiría el manejo de reutilización de código.
En [7] los autores utilizan *Multiparty Session Types* [6] para especificar la comunicación global de un protocolo de varios agentes. El tipo global es el modelo abstracto del sistema. Dentro de este modelo se encuentra el Contexto que especifica las constantes necesarias para representar los estados del tipo global. Los eventos que se crean en este modelo corresponden a los sistemas de transición, en otras palabras, todo el modelo abstracto representa la “coreografía" que los agentes deben de seguir. Los tipos locales vienen a extender el modelo previamente definido, es decir, este tipo viene a ser el “refinamiento" del modelo. El fin último es especificar patrones de comunicación utilizando *Session Types* para facilitar la tarea de modelado.

III. RESULTADOS

Los resultados se mostrarán de dos maneras. La primera consistirá en la especificación de la traducción desde NTCC a Event-B usando unos patrones de diseño de los modelos especificados como ejemplos de sincronización de procesos. Luego se usará un lenguaje de alto nivel que permite realizar de manera más natural la especificación de la sincronización de procesos.

Para ello se crea la definición de dependencias, que son básicamente modelos de la sincronización de procesos en los cuales se diseña el mapeo entre los lenguajes formales. Las dependencias se muestran a continuación:

1. Dependencia Simple:
   Representa la sincronización de procesos básicos. Sean A y B 2 procesos distintos, IF A THEN B indica que el proceso B sucede solo si el proceso A es ejecutado.
2. Dependencia Compuesta: And
   Representa la ejecución de un proceso dependiente de otros dos. Sean A y B dos procesos distintos, IF A AND B THEN C indica que el proceso C se ejeucta si y solo si A y B son ejecutados.
3. Dependencia Compuesta: Or:
   Realiza un condicional de ejecución de procesos con un condicional "o". La sentencia IF A OR B THEN C indica que C es ejecutado solo si A o B son ejeuctados.
4. Dependencia Futura Compuesta: And:
   Indica un condicional a futuro sobre una ejecución de procesos. La sentencia IF A THEN B AND C se refiere a que los procesos B y C dependen de la ejecución directa de A.
5. Dependencia Futura Compuesta: Or:
   Indica un condicional a futuro sobre procesos. La sentencia IF A THEN B OR C hace referencia a que una vez se ejecuta A solo puede ejecutarse B o C pero no ambos ("o-exclusivo").

Con base a las anteriores dependencias se crean los modelos de sincronización de procesos tanto en NTCC a Event-B como desde el lenguaje de alto nivel. Luego se extienden estos modelos agregándoles condiciones sobre el tiempo (variable IFF y ALWAYS…IF) con su respectiva traducción. La gramática especificada en el lenguaje de alto nivel diseñado para hacer las traducciones de NTCC a Event-B se muestran a continuación:

- Do A
- A AND B
- A OR B
- IF A THEN B
- NUM A
- ALWAYS A END

Siendo A y B procesos distintos.

IV. DISCUSIÓN Y CONCLUSIONES

La sincronización de sistemas reactivos requieren, por un lado, el uso del tiempo para conocer el orden de la ejecución de los procesos, y por otro lado, el no determinismo, para poder sincronizar sistemas como el envío de mensajes sin respuesta (tipo streamming, por ejemplo). La manera como en Event-B se realizaría esto sería creando guardas y variables que permitan levantar una *bandera* que indicara qué procesos (en este caso eventos) ejecutar. Sin embargo y como se observó a lo largo del proyecto, realizar directamente este proceso resulta muy exhaustivo por lo que debe existir otra manera de sincronizarlos. Es por esta razón que aparece el lenguaje NTCC, ya que con la noción del tiempo que posee y sus operadores para representar acciones no determinísticas permiten crear modelos de sincronización de sistemas reactivos de manera mucho más eficiente y más cómoda.

Un modelador de sistemas podrá especificar la sincronización de los procesos de una manera más sencilla, sin tener que pasar por el uso de guardas y variables que permitan esta sincronización, además de preguntarse si existe algún error en la sincronización de procesos mostrando los invariantes dentro del modelo acerca de la sincronización.

V. REFERENCIAS

[1] Frank D. Valencia Mogens Nielsen Catuscia Palamidessi. Temporal Concurrent Constraint Programming: Denotation, Logic and Applications". In: Nord. J. Comput. 9.1 (2002), pp. 145-188.

[2] Jean-Raymond Abrial. Modeling in event-b, system and software engineering (1ra ed.) Cambridge University Press, 2010.

[3] Thai Son Hoang, Andreas Furst, Jean-Raymond Abrial, "Event-B Patterns and Their Tool Support," Third IEEE International Conference on Software Engineering and Formal Methods (SEFM'05), pp. 210-219, 2009 Seventh IEEE International Conference on Software Engineering and Formal Methods, 2009.

[4] Vijay A. Saraswat. Concurrent Constraint Programming. MIT Press, 1993.

[5] Carlos Olarte, Camilo Rueda, and Frank D. Valencia. Models and emerging trends of concurrent constraint programming". En: Constraints 18.4 (2013), pp. 535-578.

[6] Pierre-Malo Denielou and Nobuko Yoshida. \Multiparty Session Types Meet Communicating Automata". In: ESOP. Ed. by Helmut Seidl. Vol. 7211. Lecture Notes in Computer Science. Springer, 2012, pp. 194-213. isbn: 978-3-642-28868-5.

[7] Carlos Olarte and Camilo Rueda. Communicating Systems in Event-B. Tech. rep. 2014. url: http:%20//cic.puj.edu.co/~caolarte/sessionB/.