UML2SC: Herramienta para el diseño de sistemas electrónicos complejos utilizando los lenguajes UML y SystemC

UML2SC: A tool for developing complex electronic systems using UML and SystemC

Blanca Alicia Correa, Juan Fernando Eusse, Danny Múnera, Silvio Sepúlveda, Juan Fernando Vélez, José Edinson Aedo*

Grupo de Microelectrónica y Control, Facultad de Ingeniería, Universidad de Antioquia Calle 67 N.° 53-108. Medellín, Colombia

(Recibido el 1 de Julio de 2008. Aceptado el 12 de marzo de 2009)

Resumen

En este artículo se presenta un proceso para la transformación de sistemas basados en hardware/software descritos mediante UML a código esqueleto SystemC. Se introduce la herramienta UML2SC, basada en librerías de Java, mediante la cual se puede realizar este proceso de transformación. La herramienta UML2SC, permite obtener código esqueleto SystemC a partir de modelos descritos en UML mediante los diagramas de clases y estructura compuesta. Los resultados de la utilización de esta herramienta se presentan mediante un ejemplo, el cual describe el modelo funcional de una CPU RISC de 16 *bits*.

----- *Palabras clave:* UML, SystemC, *hardware*, *software*, diseño a nivel de sistema, SoC

Abstract

In this paper, we propose a transformation process in which SystemC skeleton code is extracted from UML models that describe hardware/software based systems. The tool UML2SC, which is based on Java libraries, is also introduced. This tool allows the transformation of UML class and composite structure diagrams to SystemC skeleton code. An implementation example is given to illustrate the transformation process implemented in the UML2SC tool. The example describes the model of a CPU RISC of 16 bits.

----- *Keywords:* UML, SystemC, hardware, software, system-level design, SoC

* Autor de correspondencia: teléfono: + 57 + 4 + 219 55 67, correo electrónico: blanca@microe.udea.edu.co (B.A.Correa)

Introducción

Los recientes avances en la tecnología de semiconductores permiten la implementación de sistemas electrónicos muy complejos en un solo *chip*. Esto ha hecho posible el desarrollo de los denominados *System-on-Chip (SoC),* en los cuales componentes de hardware (HW) y software (SW) se integran en un solo *chip* con el fin de crear un sistema electrónico complejo. Por esta razón, recientemente se busca el desarrollo de metodologías y herramientas que permitan el diseño eficiente de sistemas electrónicos de gran complejidad [1]. En este contexto, una línea de investigación propone emplear el Lenguaje de Modelado Unificado (UML) [2], para la especificación y el diseño inicial de sistemas electrónicos complejos tales como los SoC [1], y luego realizar una transformación al lenguaje SystemC [3], con el fin de validar y verificar estos sistemas. UML es un lenguaje gráfico de modelado que emplea diferentes diagramas para la especificación de los sistemas. Este lenguaje permite que la especificación de un sistema se realice mediante notaciones estándar sin necesidad de emplear lenguaje natural, lo cual causa errores y ambigüedad. Por otra parte, SystemC es un lenguaje de programación compuesto por librerías de C++ para la descripción y la simulación de sistemas constituidos por componentes de HW y SW. Los modelos descritos mediante este lenguaje se pueden simular fácilmente, es decir, sin tener que realizar una descripción muy detallada del sistema.

Recientemente se han desarrollando estudios que buscan la transformación de modelos UML a código SystemC [4-8]. En estos trabajos no se reporta el desarrollo de herramientas de transformación que incluyan soporte para el diagrama de estructura compuesta, el cual es un nuevo diagrama de la reciente versión de UML 2.0. El diagrama de estructura compuesta permite la descripción de la estructura interna de una clase, lo cual facilita la representación de la estructura interna de un módulo en SystemC. En este artículo se presenta un proceso de diseño en el cual el sistema a diseñar se especifica inicialmente usando los diagramas de clases y estructura compuesta del lenguaje UML 2.0, y luego mediante la herramienta UML2SC se realiza la transformación a código SystemC. En este nivel se lleva a cabo la simulación y validación del sistema. La herramienta UML2SC está basada en librerías de Java. Esta herramienta parte de la especificación de un sistema en XMI (*XML Metadata Interchange*) [9] y permite generar código esqueleto SystemC. La especificación de un sistema en XMI se obtiene a partir de modelos en UML que son realizados empleando la herramienta *Enterprise Architect* (EA) [10].

Este artículo se encuentra estructurado de la siguiente forma: En la sección 2 se describen de forma detallada los diagramas UML seleccionados para el modelado de sistemas HW/SW. En la sección 3 se realiza una breve descripción de los elementos principales de SystemC. En la sección 4 se presenta el proceso de transformación de UML a SystemC y se describe la herramienta UML2SC. En la sección 5 se presenta un ejemplo en el cual se diseña un procesador RISC de 16 bits con un conjunto de instrucciones reducido con el fin de ilustrar el proceso de diseño. Por último, la sección 6, presenta las conclusiones.

UML

UML es un lenguaje gráfico de modelado para la especificación, visualización, construcción y documentación de sistemas [2]. UML permite modelar diferentes vistas de un sistema por medio de diagramas, aportando diferentes perspectivas y niveles de detalle que facilitan su comprensión. Hasta el momento, este lenguaje ha sido empleado comúnmente para el desarrollo de SW. Sin embargo, la versión más reciente de UML, UML 2.0, incluye notaciones que facilitan el modelado de sistemas basados en HW/SW. En UML 2.0 se definen trece tipos de diagramas, sin embargo en este estudio, se han seleccionado tan sólo un subconjunto de diagramas para el modelado de los sistemas basados en HW/SW y la generación de código SystemC: el diagrama de clases y el diagrama de estructura compuesta. Estos diagramas permiten describir aspectos estructurales de un sistema. El diagrama de clases permite repre-

sentar los elementos que conforman el sistema a modelar por medio de clases y las relaciones entre estas. El diagrama de estructura compuesta permite ilustrar la estructura interna de las clases por medio de partes, puertos y conectores. Por otro lado, UML puede ser adaptado para dominios de modelado específicos mediante la definición de un perfil. Un perfil de UML es un grupo de estereotipos, restricciones y valores etiquetados que adicionan información específica de dominio al UML. Para el proceso de transformación de UML a SystemC propuesto en este artículo se han aplicado los estereotipos que se muestran en la tabla 1 a los modelos en UML, los cuales se basan en el perfil de UML 2.0 para SystemC presentado en la referencia [11]. En la tabla 1 también se ilustran las metaclases relacionadas con los estereotipos.

Tabla 1 Metaclases relacionadas con los estereotipos

Estereotipo	*Metaclase UML*
sc_module	Clase
sc_channel	Clase
sc_prim_ channel	Clase
sc_interface	Interfaz
sc_port	Puerto
sc_export	Puerto
sc_connector	Conector
sc_method	Operación/Máquina de estado
sc_thread	Operación/Máquina de estado

SystemC

SystemC consiste en una colección de clases codificadas en C++ para el diseño, la simulación y la verificación de sistemas basados en HW y SW, e incluye un simulador basado en eventos [3]. SystemC permite modelar y verificar los diseños en un alto nivel de abstracción [12], en las primeras etapas del proceso de diseño para luego refinar estos diseños incorporando detalles de implementación, lo cual eventualmente conlleva a los prototipos y al diseño completo de un SoC. SystemC separa la computación de la comunicación con el fin de ofrecer una solución simple y modificable a los diseñadores que buscan técnicas más eficientes de modelado y simulación [13]. A continuación se describen los elementos principales de SystemC:

- *Módulo:* Un módulo es la unidad básica en un diseño en SystemC. Una aplicación típica en SystemC consiste normalmente de una jerarquía de módulos.
- *Proceso:* Un proceso permite describir funcionalidad y concurrencia, y se encuentra contenido en los módulos como una función miembro especial. Existen tres tipos de procesos: *SC_THREAD*, *SC_CTHREAD* y *SC_ METHOD*.
- *Evento:* Un evento define si la ejecución de un proceso debe dispararse o suspenderse y el momento en que esto debe ocurrir.
- *Puerto:* Un puerto permite que los módulos se comuniquen con su ambiente externo.
- *Export:* Un *export* permite que un módulo pueda proveer una interfaz a un módulo de mayor jerarquía mediante la definición de un conjunto de servicios.
- *Interfaz:* Una interfaz define un conjunto de operaciones que un canal debe implementar, los cuales son accedidos por los puertos de un módulo.
- *Canal:* Los canales permiten el proceso de partición entre comunicación y computación en SystemC ya que un canal implementa las interfaces, pero no las define. En SystemC existen dos tipos de canales: primitivos y jerárquicos. Un canal primitivo no contiene estructura interna y permite la comunicación por medio de los métodos que hereda. En cambio, un canal jerárquico es básicamente un módulo que puede tener estructura y procesos internos.

Proceso de transformación de UML a SystemC y herramienta UML2SC

El proceso de transformación de UML a SystemC se lleva a cabo de la siguiente forma:

1. Se modela un sistema empleando la notación UML usando la herramienta EA.
2. Se genera automáticamente el código XMI correspondiente al modelo descrito en UML, empleando la misma herramienta de modelado para UML.
3. Se emplea un *parser* que permite organizar el código XMI en una estructura de datos.
4. Se combina la estructura de datos obtenida tras el proceso de *parsing*, con plantillas de SystemC empleando un motor de plantillas (*template engine*).

En la figura 1 se puede apreciar el proceso de transformación de UML a código SystemC.

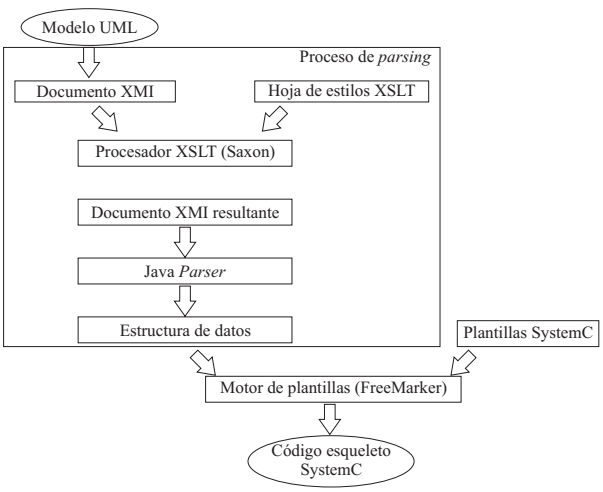

Figura 1 Proceso de transformación de UML a SystemC

A continuación se explican detalladamente cada uno de los pasos involucrados en este proceso de transformación.

Proceso de parsing

El proceso de *parsing* está conformado por dos etapas principales como se muestra en la figura 2, la primera es la aplicación de una hoja de estilos XSLT mediante el paquete *Saxon* [14], la segunda parte es la obtención de la estructura de datos mediante el *parser Document Object Model* (DOM) [15]. A continuación se explican estas etapas:

1. Aplicación de una hoja de estilos XSLT: El XSLT (*eXtensible Stylesheet Language (XSL) transformations*) [16] es usado para transformar un documento XML [17] en otro documento XML, o cualquier otro tipo de documento. Con XSLT se puede adicionar o remover elementos y atributos en el archivo de salida, incluso se pueden cambiar de lugar u ordenar elementos realizando pruebas y tomando decisiones acerca de cada elemento para eliminarlo o adicionarlo en el archivo de salida. Por tanto, en esta etapa del proceso de *parsing* se toma el documento XMI generado por la herramienta EA a partir del modelo del sistema que se quiere simular y se elimina la información irrelevante o redundante de éste. Además se reordenan los datos seleccionados, y se unifican y organizan las etiquetas XML del documento resultante facilitando la creación de la estructura de datos en la siguiente etapa.

   Esta etapa del proceso de *parsing* se basa en la utilización del procesador XSLT de código abierto *Saxon-B* en su versión para Java. Mediante esta herramienta se aplica una hoja de estilos XSLT al documento obtenido con el EA para obtener al final un documento XMI con la información relevante del proceso.

2. Obtención de la estructura de datos: Para la obtención de la estructura de datos se emplea el *parser* DOM. Un *parser* transforma un texto de entrada en una estructura de datos, la cual usualmente es una estructura de tipo árbol. Esta estructura de datos captura la jerarquía de la entrada y resulta adecuada para realizar un procesamiento posterior de los datos. El DOM, es una forma de representar los elementos de un documento estructurado (tal como una página *web* HTML o un documento XML) como objetos que tienen sus propios métodos y propiedades.

La parte crucial del proceso de *parsing* es el *Application Programming Interface* (API) de Java DOM, el cual permite cargar un documento XML como un objeto en Java con una estructura de árbol abstracto. Luego se navega por esta estructura organizando los datos en arreglos de clases predefinidas que sirven como modelo de datos en la siguiente etapa del proceso de transformación.

En esta etapa es de vital importancia definir con precisión cuales son los parámetros que debe tener cada clase o interfaz definida en el modelo del sistema con el fin de tener la información necesaria a la hora de realizar la transformación a código SystemC.

Los parámetros que se toman en cuenta para definir un módulo de SystemC son: variables, herencia (tanto de interfaces como de clases), submódulos definidos, operaciones, puertos y canales. En el caso de las interfaces sólo se definen sus métodos y su herencia.

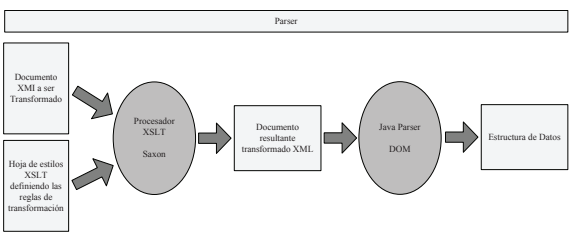

Figura 2 Etapas en el proceso de *parsing*

Motor de plantillas

Un motor de plantillas combina una o más plantillas con un modelo de datos para producir uno o más documentos resultantes. El motor de plantillas seleccionado para la herramienta UML2SC es *Freemarker* [18]. Este motor de plantillas se basa en la idea de realizar transformaciones sobre código en un lenguaje de programación determinado, en este caso Java, hacia otros lenguajes como son HTML, XML o incluso texto plano, y se compone de tres partes principales:

1. El modelo de datos: Son los datos completamente estructurados dentro del lenguaje de programación origen que serán luego transformados en la salida del motor de plantillas.
2. La plantilla: Es el molde que le permite al motor de plantillas realizar la transformación del modelo de datos en la salida, de una forma consistente y coherente.
3. La salida: Es el resultado del proceso de transformación que se da al combinar el modelo de datos con la plantilla a través del motor de plantillas.

La herramienta UML2SC puede ser utilizada a través de la página *web* del proyecto [19]. La figura 3 muestra la interfaz *web* para la generación de código:

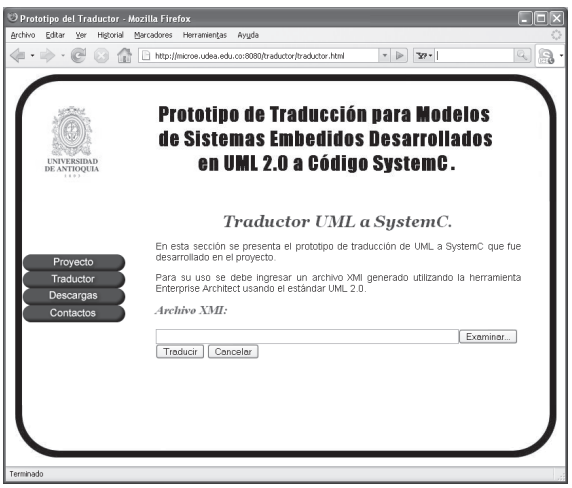

Figura 3 Interfaz *web* para generación de código SystemC

Estudio de caso

Se decide elaborar un estudio de caso real basado en experiencias previas de equipos investigativos en las cuales la falta de verificación llevó al no funcionamiento de la implementación inicial y a la necesidad de un proceso de re-diseño tanto del software como del hardware. Se analizó el proceso de desarrollo de una CPU de 16 bits y su integración a un SoC enfocado hacia aplicaciones agroindustriales [20]. Dentro de [20] se encuentra que además de no tener una metodología de validación funcional del sistema como un todo (HW + SW), durante la implementación de la especificación propuesta para el SoC no se evaluó el grado de validez conceptual del mismo, incurriendo en un error metodológico fundamental.

Según [20], el *datapath* de la CPU de 16 bits y el diagrama de bloques del SoC son los mostrados en las figuras 4 y 5:

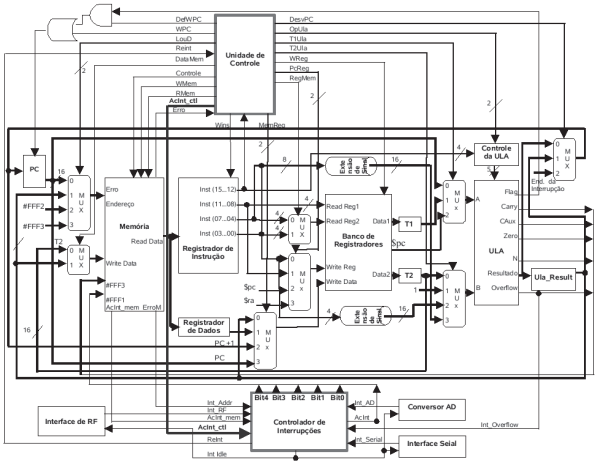

Figura 4 *Datapath* para la CPU propuesta por [20]

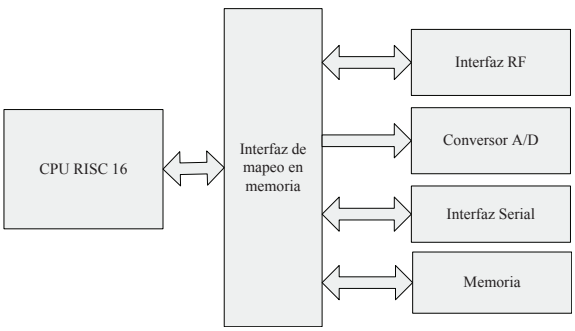

Figura 5 Diagrama de bloques SoC de irrigación propuesto en [20]

Para la aplicación de nuestra metodología, se propone un modelado puramente funcional de la CPU utilizando *Transaction Level Modeling 2.0* (TLM 2.0) [12], y su concepto de *socket* para modelar la comunicación entre todos los módulos del SoC en un alto nivel de abstracción. Debido a que el modelo propuesto en este artículo es orientado únicamente a la verificación funcional, se elimina la necesidad de la interfaz de mapeo en memoria y se analizan las comunicaciones entre los módulos como punto a punto, esto con el fin de separar el procesamiento de la comunicación y simplificar la descripción sin perder precisión al modelar el sistema. El diagrama de estructura compuesta para el sistema TLM 2.0 propuesto es el mostrado en la figura 6:

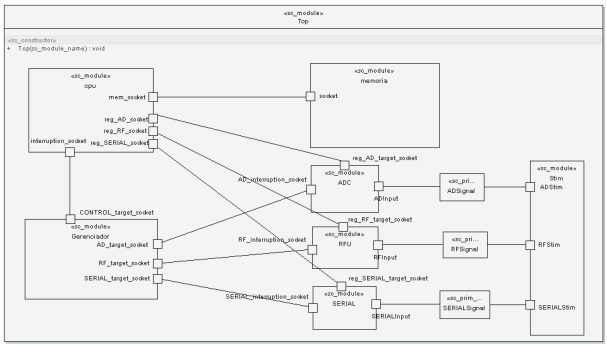

Figura 6 Diagrama de estructura compuesta propuesto para el modelo

De la figura anterior se puede observar que dividimos la CPU para separar procesamiento de comunicación, separando de ella la unidad de administración de interrupciones del resto del procesador. Además se puede observar los *sockets* TLM2.0 (tanto iniciadores como objetivos), los cuales pertenecen a la clase tlm::tlm_b_initiator_socket<> y tlm::tlm_b_target_socket<> respectivamente. Luego de plantear el diagrama de bloques del modelo, se procedió a realizar el modelado del SoC mediante la herramienta EA, para luego realizar el proceso de traducción del mismo. El modelo UML de la CPU se presenta en la figura 7, se omiten los modelos de los elementos restantes del SoC por consideraciones de espacio y por este estar disponible en la página del proyecto [19]. Luego de realizar el modelado en UML del SoC, se procede a la generación del código SystemC para el mismo.

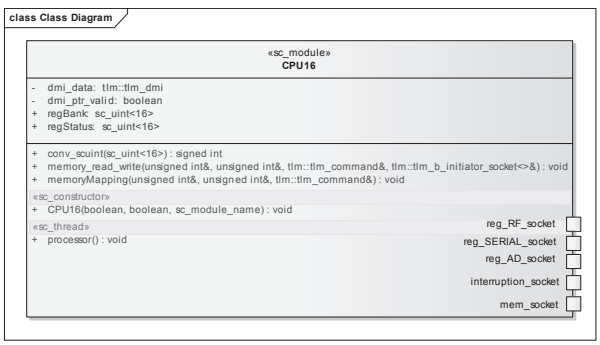

Figura 7 Diagrama de clases de la CPU

A continuación se muestran fragmentos del código generado para la CPU, los otros módulos son omitidos.

```
#include "systemc.h"
#include "tlm_bw_b_transport_if.h"
class CPU16:sc_module ,tlm_bw_b_transport_if{
public:
  //MODULE PORT DECLARATION - Templated
  tlm::tlm_b_initiator_socket<> interruption_socket;
  tlm::tlm_b_initiator_socket<><> mem_socket;
  tlm::tlm_b_initiator_socket<><> reg_AD_socket;
  tlm::tlm_b_initiator_socket<><> reg_RF_socket;
  tlm::tlm_b_initiator_socket<><> reg_SERIAL_socket;

    //MODULE'S LOCAL CHANNEL DECLARATIONS - Templated
    //No associated local channels

    //SUB-MODULE DECLARATION - Templated
    //There are not sub-modules to translate

    //PROCESS DECLARATION, THE PROCESS REGISTRATION IS DONE IN THE //MODULE
    CONSTRUCTOR
     public: signed int conv_scuint(sc_uint<16> conv){…}

    public: void memory_read_write(unsigned int& addr,unsigned int&
    data,tlm::tlm_command& cmd,tlm::tlm_b_initiator_socket<>& ini_soc-
    ket){…}

    public: void memoryMapping(unsigned int& direccion,unsigned int&
    data,tlm::tlm_command& cmd){…}

    public: void processor(){…}

    //INTERFACE INHERITED METHODS
    public: virtual void invalidate_direct_mem_ptr(sc_dt::uint64 end_
    range,sc_dt::uint64 start_range){}

    //The SC_HAS_PROCESS statement tells the c++ compiler to expect //for a
    constructor - emplate
    SC_HAS_PROCESS(CPU16);

    //Now here it is the class constructor - Templated
    CPU16(sc_module_name module_name,bool _pasos,bool _debug):sc_
    module(module_name),_pasos(_pasos),_debug(_debug){
        //THIS IS THE CONSTRUCTOR BODY, IN HERE WE DECLARE THE //PROCESS
        REGISTRATION, THE SUB-MODULE IMPLEMENTATION
        //THE CHANNEL IMPLEMENTATION AND THE INTER-MODULE BINDING
```

```
        SC_THREAD(processor); //PROCESS REGISTRATION – Templated
    }//Constructor ends here

    //Constructor parameters and other module member variables
    private:
    //MEMBER VARIABLES
    tlm::tlm_dmi dmi_data;
    bool dmi_ptr_valid;
    sc_uint<16> regBank;
    sc_uint<16> regStatus;

    //CONSTRUCTOR PARAMETERS
    bool _pasos;
    bool _debug;
};
```

Luego de la generación del código esqueleto del sistema, el equipo de trabajo procedió a implementar la funcionalidad del SoC para luego verificarlo mediante la ejecución de programas de software de adquisición de señales del conversor A/D hacia memoria, organización de matrices y generación de secuencias. Durante el flujo del proyecto se detectó que la serie de instrucciones planteada por la especificación inicial no era apto para el tipo de aplicación que se pretendía desarrollar con el SoC, al carecer de instrucciones de multiplicación y división de enteros. Se comprueba entonces la utilidad de la metodología para el diseño y validación de sistemas en silicio.

Conclusiones

En este artículo se presentó el proceso para la transformación de notaciones en UML al lenguaje SystemC soportado por la herramienta UML2SC. La herramienta UML2SC permite la transformación de diagramas de clases y de estructura compuesta a código esqueleto SystemC. La selección del modelo de datos es vital en el proceso de traducción, pues, en caso de que el modelo implementado no sea adecuado, el proceso de traducción será deficiente ya que no contará con todos los datos necesarios para el proceso. Al aplicar el proceso de transformación para el modelado funcional de una CPU RISC de 16 *bits* se comprobó que la metodología es útil para el diseño y validación de sistemas *On-Chip*, ya que permite detectar las fallas del proyecto en una etapa temprana del mismo, donde se minimizan los costos de rediseñar el sistema y se pueden corregir dichas fallas disminuyendo el impacto sobre el desarrollo del proyecto. Como trabajo futuro se propone la generación de código SystemC a partir del diagrama de estados de UML para la descripción del comportamiento de los sistemas y de los diagramas de secuencias y de tiempos para la obtención de *test-benches* en SystemC.

Referencias

1. G. Martin, W. Müller. *UML for SoC Design*. Springer. Netherlands. 2005. pp 17-36.
2. OMG, *Unified Modeling Language: Superstructure, version 2.0, formal/07-03-05*. Disponible *online*: http://www.uml.org. Consultada el 25 de mayo de 2008.
3. SystemC home page. Disponible *online*: http://www.systemc.org. Consultada el 14 de febrero de 2008.
4. V. Sinha, F. Doucet, C. Siska, R. Gupta, S. Liao, A. Ghosh. "YAML: A tool for hardware design visualization and capture". *Proc of the 13th international symposium on System Synthesis*. 2000. pp. 9.
5. K. Nguyen, Z. Sun, P. Thiagarajan, W.F. Wong. "Model-driven SoC design via executable UML to SystemC". *Proc. of RTSS*. 2004. pp. 459.
6. W. Tan, P. Thiagarajan, W. Wong, Y. Zhu, S. Pilakkat. "Synthesizable SystemC code from UML models". *International Workshop on UML for SoC Design*. 2004. pp. 1-3.

7. C. Xi, L. J. Hua, Z. Cheng, S. Y. Hui. "Modeling SystemC design in UML and automatic code generation". *Proc. of ASP-DAC*. 2005. pp. 932.

8. Y. Wang, X. Zhou, B. Zhou, L. Liang, C. Peng. "A MDA based SoC Modeling Approach using UML and SystemC". *Proc. of CIT*. 2006. pp. 245.

9. Object Management Group. XML Metadata Interchange (XMI). Disponible online: http://www.omg.org/technology/documents/formal/xmi.htm. Consultada el 18 de julio de 2008.

10. Enterprise Architect. Disponible *online*: http://www.sparxsystems.com. Consultada el 25 de julio de 2008.

11. E. Riccobene, P. Scandurra, A. Rosti, S. Bocchio. "A UML 2.0 Profile for SystemC: Toward High level SoC Design". *Proc. of the 5th ACM international conference on Embedded Software (EMSOFT'05)*. 2005. pp. 138 - 141.

12. F. Ghenassia, *Transaction-Level Modeling with SystemC*. Ed. Springer. Netherlands. 2005. pp 10-14.

13. T. Grötker, S. Liao, G. Martin, S. Swan. *System Design with SystemC*. Ed. Kluwer Academic Publishers. New York. 2004. pp. 1-3.

14. Saxon. Disponible *online*: http://saxon.sourceforge.net/ Consultada el 20 de junio de 2008.

15. Java DOM. Disponible *online*: http://www.jdom.org. Consultada el 27 de mayo de 2008.

16. XSL Transformations (XSLT). Disponible *online*: http://www.w3.org/TR/xslt Consultada el 7 de junio de 2008.

17. Extensible Markup Language (XML). Disponible *online*: http://www.w3.org/XML. Consultada el 17 de abril de 2008.

18. FreeMarker. Disponible *online*: http://www.freemarker.org. Consultada el 12 de junio de 2008.

19. Página Proyecto UML2SC. Disponible *online:* http://microe.udea.edu.co:8080/traductor. Consultada el 23 de julio de 2008.

20. J. Domingues, *Implementaçao de um Processador RISC 16-bits CMOS Num Sistema em Chip*. Tesis de Mastría en Ingeniería Eléctrica. Facultad de Tecnología. Universidad de Brasilia (UnB). Brasil. 2004.