La Tubería de Diseño de Sistemas Integrados

Javier Iparraguirre y Claudio Delrieux

Departamento de Ing. Eléctrica y Computadoras
Universidad Nacional del Sur - Alem 1253 (8000) Bahía Blanca
javierip@ieee.org, claudio@acm.org

Abstract **-** A partir de la problemática existente en el diseño de Sistemas Integrados (*embedded systems*) se propone en este trabajo un método formal de diseño. Se describen las tareas involucradas y se presenta un marco de trabajo que minimiza los recursos a utilizar. A partir de un dispositivo conocido, se presenta un ejemplo de aplicación. Finalmente, se propone un agente de software que asiste a los diseñadores durante el proceso.

Keywords **-** Arquitectura de computadoras. Sistemas Integrados. Ingeniería del Software. Sistemas de tiempo real.

I. Introducción

Se considera un *sistema integrado* (SI) a aquel que contiene hardware y software dedicado a una tarea específica. La concepción de estos diseños apunta a resolver problemas particulares, dejando en un segundo plano la versatilidad de aplicación. En cuestión de unos pocos años se produjo una gran revolución en este área. Entre otros, el disparador de esta explosión tecnológica se debe a la telefonía celular. Las compañías fabricantes de estos dispositivos, conscientes de que estos productos son plataformas para aplicaciones mucho más versátiles que lo que ellos imaginaban, constantemente buscan la manera de captar el ingenio de los programadores creativos por medio de premios, concursos o proyectos free-lance. Actualmente se presentan muchas oportunidades para desarrollar e implementar aplicaciones integradas. La sociedad acepta sin problemas los desarrollos en ese sentido (en la medida en que sus prestaciones sean transparentes) y cada vez hay más soluciones a problemas no resueltos por otras tecnologías.

Los fabricantes de componentes electrónicos ofrecen gran cantidad de dispositivos que satisfacen las demandas del mercado de los SI. Generalmente se pueden encontrar varias alternativas de desarrollo para un mismo problema. Este panorama, que en primera instancia parece alentador, puede tornarse problemático a la hora de decidir qué herramienta utilizar. Como ejemplo se puede citar el caso de los microprocesadores, los microcontroladores y los DSP. En el campo del software se presenta una situación similar. Por lo tanto, el diseño de un SI es un proceso en el cual influyen varios factores, tanto del hardware como del software.

Otra característica única de los SI es que abarcan al menos dos áreas del conocimiento tales como la ingeniería electrónica y la ingeniería en computación. Generalmente, las personas que diseñan aplicaciones no tienen una formación completa en las dos disciplinas. Es decir, estudiaron alguna de ambas ingenierías y se ven forzadas a solucionar problemas que tienen algún aspecto sobre el cual poco conocen. Si a este escenario se le suma la necesidad de minimizar el tiempo de desarrollo, se

obtiene un complejo panorama lleno de obstáculos. En consecuencia, es muy probable encontrarse con callejones sin salida a la hora de transitar por las rutas del diseño de los SI.

II. Características Indispensables de un SI

A continuación se mencionan las características indispensables de un SI y los problemas asociados más frecuentes en el proceso de diseño:

- **Elección del hardware.** Los recursos físicos del sistema pueden no ser suficientes para satisfacer los requerimientos de la aplicación. Si se dispone de excesivos recursos se presenta un error de diseño también dado que el consumo innecesario de energía es contraproducente.
- **Elección del software.** Además de cumplir con los requerimientos de diseño, el software debe ser diseñado de manera tal que sea posible el mantenimiento, además de poder ampliarlo, escalarlo y reutilizarlo en futuras aplicaciones, tanto para cambios en las prestaciones de la aplicación, como para migraciones de la misma a plataformas diferentes y/o cambios en el hardware subyacente.
- **Compatibilidad.** La mayoría de los SI se comunican con otras plataformas de computación por medio de algún medio físico de comunicación con su protocolo asociado. En muchos casos se desarrollan sistemas de comunicación de manera unilateral, impidiendo la comunicación con plataformas existentes en el mercado.
- **Escalabilidad.** Una alternativa a tener en cuenta en tiempo de diseño es la concepción de escalabilidad. Los SI pueden diseñarse de manera tal que puedan ser ampliados a medida que los requerimientos exigen prestaciones superiores.
- **Reuso.** Algunos de los problemas que se pretenden resolver son “similares” a otros previamente resueltos. Los diseñadores pueden utilizar desarrollos “antiguos” en los actuales con el fin de minimizar el uso de recursos utilizados.
- **Practicidad.** Gran número de desarrollos no son exitosos debido a la excesiva complejidad de una o varias de sus partes. Aunque no lo parezca, la simplicidad es un factor clave en cualquier diseño.
- **Costo adecuado.** A pesar de cumplir con los requerimientos, un sistema no está correctamente diseñado si cuesta más de lo que los usuarios finales pueden o están dispuestos a pagar.
- **Tiempo de desarrollo.** El tiempo al mercado debe minimizarse en todo proyecto de diseño de un SI. En la actualidad existe un gran número fabricantes o diseñadores que se encuentran desarrollando distintos sistemas con el fin de solucionar los mismos problemas.

III. Un método de diseño para SI

En este trabajo se propone un método de diseño de sistemas integrados. Se parte desde los requerimientos hasta implementar el código del software y el hardware subyacente. Minimiza la intervención de los diseñadores de manera tal que optimiza los recursos a utilizar. Se basa en una línea de desarrollo en la cual los requerimientos abstractos generan nuevas instancias de diseño y finalmente se reduce a tareas de implementaciones prácticas de problemas abstractos. Permite automatizar la construcción de las bases de diseño dejando los detalles para los desarrolladores. Se

puede decir que las ventajas del método se basan en ordenar, minimizar recursos, minimizar errores, automatizar tareas y permitir el trabajo en varios grupos o equipos.

El método propuesto presenta un marco de trabajo el cual ordena todas las tareas a realizar durante todo el tiempo de desarrollo. Permite obtener un sistema con recursos balanceados a las demandas y optimiza el cumplimiento de los requerimientos. Está basado en las herramientas propias de la ingeniería en software y agrega nuevos conceptos aplicables a los SI. Se propone un diagrama de flujo en el cual se muestran los procesos involucrados en la construcción al cual se lo denomina La Tubería de Diseño. La Tubería fue desarrollada a partir el Diagrama de Ciclo de Vida adaptado a sistemas integrados desarrollado en trabajos previos. Este esquema presenta dos ventajas principales. La primera consta el la posibilidad de mostrar la relación de los procesos. La segunda consta en permitir analizar la interrelación de tareas.

Ciclo de Vida

El ciclo de vida consta de 12 etapas (ver Fig. 1). Cada una de ellas fue nombrada con una letra del alfabeto con el fin de simplificar el esquema. En el borde inferior del diagrama se puede ver una indicación de tiempos de ejecución. Este caso no es representativo de los tiempos reales, ya que cada etapa tiene sus propios tiempos de duración. Cada una de las etapas tiene un tiempo de duración particular, al cual se lo denomina Txi, donde X representa la etapa en particular. Se denomina a T1 como tiempo de diseño y a T2 como tiempo de implementación.

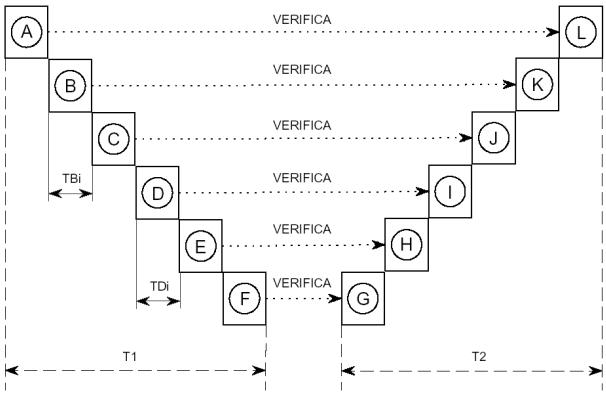

Figura 1: Ciclo de vida

A continuación se detalla el nombre de cada una de las etapas:

A. Requerimientos generales.
B. Requerimientos del hardware.
C. Requerimientos del software.
D. Análisis técnico - económico.
E. Construcción del hardware.
F. Construcción del software.
G. Pruebas del software.
H. Pruebas del hardware.
I. Verificación del costo de fabricación.
J. Integración y pruebas en el laboratorio.
K. Montaje y pruebas en el campo.
L. Control de calidad, operación y mantenimiento.

La Tubería de Diseño

Con el fin de organizar los *tipos* de tareas a realizar, La Tubería se divide en siete etapas o tramos (ver Fig. 2). Estos son: Requerimientos, Abstracción, Implementación, Construcción, Pruebas Individuales, Pruebas en Laboratorio y Pruebas en Campo. Cada uno de los tramos está asociado a las tareas particulares a realizar en el diseño.

El primer tramo, Requerimientos, es el menos técnico de todos. Sin embargo, es incluido dentro del proceso de diseño ya que es de suma importancia. En esta instancia se pacta con el usuario final los requerimientos del sistema a diseñar. Es fundamental comprender lo más completamente posible las tareas a realizar por el sistema debido a que una mala interpretación en esta instancia se arrastra durante todo el diseño. La tarea denominada con el numero 1 consta en acordar con el usuario final las características de diseño.

Una vez concluido el primero de los tramos, se pasa a la Abstracción, denominado con el número 2. Se propone la arquitectura general del sistema a nivel hardware y software de manear abstracta. La descripción funcional del proyecto debe ser completa. Para cumplir con esta tarea se diseñó el diagrama UMLX, *Abstract Unified Modelling Language*. El uso del mismo permite modelar cualquier SI en términos de *clases abstractas*. El diagrama consta de cuatro campos principales: Nombre, Componentes, Interfaces y Responsabilidades. A su vez, los campos Componentes e Interfases se subdividen en hardware y software (Fig. 3). En el diagrama UMLH se debe describir al sistema de manea general, sin entrar en detalles específicos de la arquitectura.

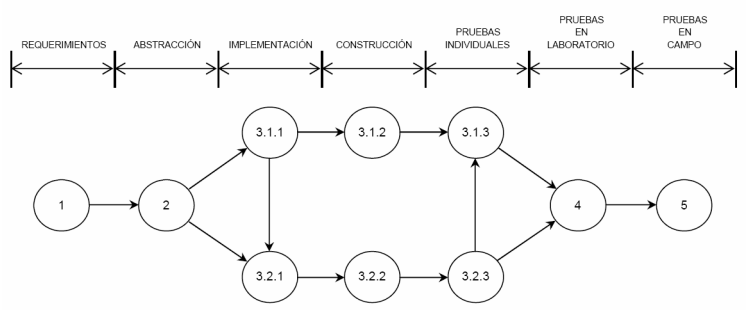

Figura 2: Tubería de Diseño

DIAGRAMA UMLX	
NOMBRE:	
COMPONENTES	HARDWARE:
	SOFTWARE:
INTERFASES	HARDWARE:
	SOFTWARE:
RESPONSABILIDADES:	

Figura 3: Diagrama UMLX

El tramo Implementación consta de dos tareas denominadas con los números 3.1.1 y 3.2.1. En ambos casos se crean *clases implementables* las cuales heredan las propiedades de las clases creadas en el tramo anterior. Estas clases modelan objetos que pertenecen al mundo del hardware o el software. Como en la etapa previa, se crearon dos diagramas: UMLX -*Abstract Unified Modelling*

Language- y UMLS -*Software Unified Modelling Language*-. Con la ayuda de estas herramientas se modela al sistema hasta el mínimo detalle. La las clases generadas heredan los atributos y métodos de sus ancestros. Se respetan los mismos conceptos generados en la programación orientada a objetos.

En el punto 3.1.1 se definen las clases implementables que modelan al hardware del sistema. Se generan los diagramas UMLH necesarios. Luego, en el punto 3.2.1 se generan los diagramas UMLS. Nótese que en la tubería se muestra claramente la interrelación entre estos dos procesos. Se debe definir el hardware para poder diseñar el software.

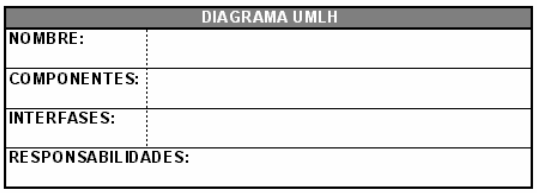

Figura 3: Diagrama UMLH

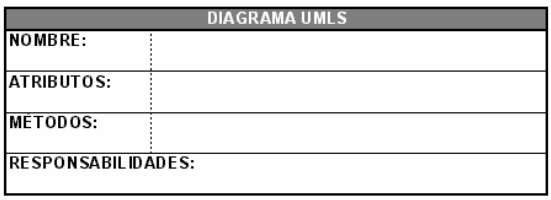

Figura 4: Diagrama UMLS

Las tareas 3.1.2 y 3.2.2 pertenecen a la tramo Construcción. La primera tarea mencionada hace referencia a la construcción de hardware y la segunda al software. En ambos casos se construyen los objetos modelados por las clases implementables. Para el caso de los objetos software el proceso de construcción es bien conocido. Sin embargo, para los objetos hardware, esta tarea es novedosa. Un objeto hardware puede ser un componente simple, como una compuerta lógica, o un conjuntos de subsistemas, como un microprocesador. La complejidad del objeto no es un limitante para la construcción.

El proceso de construcción de los objetos puede optimizarse a través de algoritmos de generación de código en un lenguaje de programación a partir de los diagramas UML. Nuevamente, esta tarea resulta familiar para el caso de la programación orientada a objetos. Como se explicará en las siguientes secciones, para el caso de los circuitos físicos este pasaje es posible gracias a la disponibilidad de lenguajes de descripción de hardware. Esta característica del método es uno de sus puntos fuertes debido a que presenta un esquema una elegante estructura formal que permite *automatizar* el diseño de SI. En las siguientes secciones se desarrolla con mayor detalle este punto.

El tramo de Pruebas Individuales consta en diseñar una estrategia de prueba para cada subsistema del diseño. Las pruebas se pueden detallar hasta el punte en el cual se ensaye el comportamiento de cada uno de los objetos. En la tubería se muestra una interrelación entre las pruebas del hardware, tarea 3.1.3, y las pruebas del software, tarea 3.2.3. Básicamente, se propone probar los primero los objetos del software para luego utilizarlos en las pruebas de los objetos físicos.

La tarea 4, incluida en el tramo Pruebas en Laboratorio, consta en probar el comportamiento del sistema ensamblado. La mayoría de los SI trabajan en ambientes poco amigables. Por lo tanto, es crucial que el proceso de diseño concluya con las Pruebas en Campo, englobadas en la tarea 5. La

estrategia de pruebas se determina a partir de los diagramas UMLX. El riesgo de omitir una prueba crucial se minimiza ya que la funcionalidad del sistema se basará en las clases abstractas, por lo tanto, las pruebas a realizar quedan *formalmente* definidas. Este es otro punto sobresaliente del método.

Construcción de Objetos

La construcción de los objetos se basa en las clases implementables. Como se mencionó anteriormente, se pueden generar algoritmos que construyan a los objetos a partir de las clases. Para el caso del software estos procedimientos son conocidos con el nombre de constructores. En el caso del hardware el mismo concepto es viable a partir de los lenguajes de descripción de hardware. A partir de la existencia en el mercado de los circuitos lógicos programables, gran parte de los componentes se pueden implementar por esta vía. En esta sección se explicará el pasaje de UMLH a VHDL.

Las siglas VHDL provienen de "VHSIC Hardware Description Lenguaje" y a su vez VHSIC quiere decir "Very High Speed Integrated Circuit". El VHDL nació como un lenguaje de modelado y documentación de sistemas electrónicos digitales. Se estandarizó mediante el estándar 1076 del IEEE en 1987 (VHDL-87). Este estándar fue extendido y modificado en 1993 (VHDL-93) y 2002 (VHDL-2002). Permite modelar sistemas digitales de manera jerárquica, definiendo de manera precisa las interfases de cada elemento.

Con VHDL, un módulo o componente de hardware se modela en dos secciones. Una es la interfaz del componente, denominado *entidad* y la otra es la *arquitectura* que describe su funcionamiento interno. Este modelo permite esconder los detalles internos de implementación e incluso definir varias implementaciones para un componente sin afectar la interfase externa. En la Figura 5 y 6 se puede ver la definición de una compuerta NOT en VHDL.

```
entity compuerta is
  port(
  Entrada : IN std_logic;
  Salida : OUT std_logic;
  );
  end compuerta;
```

Figura 5: Interfaz del componente `compuerta`.

```
architecture Comportamiento of compuerta is
  signal interna : std_logic;
  begin
    interna <= Entrada;
    Salida <= not interna;
  end architecture Comportamiento;
```

Figura 6: Arquitectura del componente `compuerta`.

El pasaje de UMLH a VHDL no presenta grandes dificultades. Es totalmente directo. Las señales listadas en el campo **Interfaces** del diagrama ULMH se asocian a las señales definidas en la estructura **Entity** en VHDL. Los **Componentes** incluidos en el diagrama UMLH se definen en el bloque **Architecture** para VHDL. La tarea de "traducción" se puede programar fácilmente, dejándole solamente a los diseñadores la definición de la estructura interna de los componentes. Un punto importante a tener en cuenta es que VHDL permite simular los circuitos antes de ser sintetizados. Por lo tanto, se puede acelerar el proceso de pruebas.

Los diseñadores deben elegir cuales de los componentes de la plataforma física serán implementados en lógicas programables y cuales no. La herramienta presentada en esta sección permite presentar varios escenarios de diseño del hardware. Logrando presentar múltiples escenarios de diseño ante un conjunto particular de clases implementables. Este es otro punto fuerte del método ya que presenta un marco de trabajo formal para probar diversas alternativas tecnológicas prácticas.

IV. Ejemplo de aplicación

Con el fin de mostrar cómo funciona el método en la práctica se decidió modelar el desarrollo completo de una aplicación para un SI existente. El hecho de disponer con la documentación del sistema elegido permite concentrarse en el método y no en la arquitectura. La plataforma en cuestión es un servidor integrado llamado TINI (Tiny InterNet Interface) fabricado por Dallas Semiconductor. A grandes rasgos, el sistema está integrado en una placa (System on a Board) e implementa una porción funcionalmente importante de UNÍS, diversos protocolos de comunicaciones, y ambiente de operación Java. La documentación pertinente puede encontrarse en http://www.ibutton.com/TINI/.

En primera instancia se crea la clase abstracta TINI, Fig. 7, la cual se representa en un diagrama UMLX. TINI será la clase madre de todo el proyecto y de esta todas las clases que se creen heredarán las propiedades. A partir de esta clase principal, se genera una estructura jerárquica de subclases. Estas nuevas clases deben modelarse siguiendo las reglas establecidas.

DIAGRAMA UMLX		
NOMBRE:	TNI	
COMPONENTES	**HARDWARE:**	Microcontrolador. Memoria ROM. Memoria RAM. Reloj de tiempo real. Controlador ethernet Sub-sistema de comunicación asincrónica. Sub-sistema de comunicación sincrónica.
	SOFTWARE:	Sistema operativo (SLUSH) Capa de interfaz de metodos nativos. Maquina virtual Java (JVM).
INTERFASES	**HARDWARE:**	Conexión RS-232. Conexión CAN. Conexión 1-Wire. conexión ethernet 10 Base-T Conexión serie sincrónico de 2 conductores. Conexión paralelo.
	SOFTWARE:	javax.comm (Comunicación RS-232) com.dalsemi.onewire (Comunicación 1-wire) com.dalsemi.comm (Comunicación CAN) com.dalsemi.shell (FTP, Telnet) java.net (TCP/IP)
RESPONSABILIDADES: Servidor de red integrado con un ambiente de operación Java.		

Figura 7: Clase TINI

En la Figura 8 se muestran las subclases que se generan a partir de la clase TINI. Puede verse la familia de clases implementables creadas. Se modela tanto el hardware como el software. Nótese que no se han definido los detalles prácticos de implementación aún. Esta característica permite mantenerse en el plano conceptual independientemente del práctico.

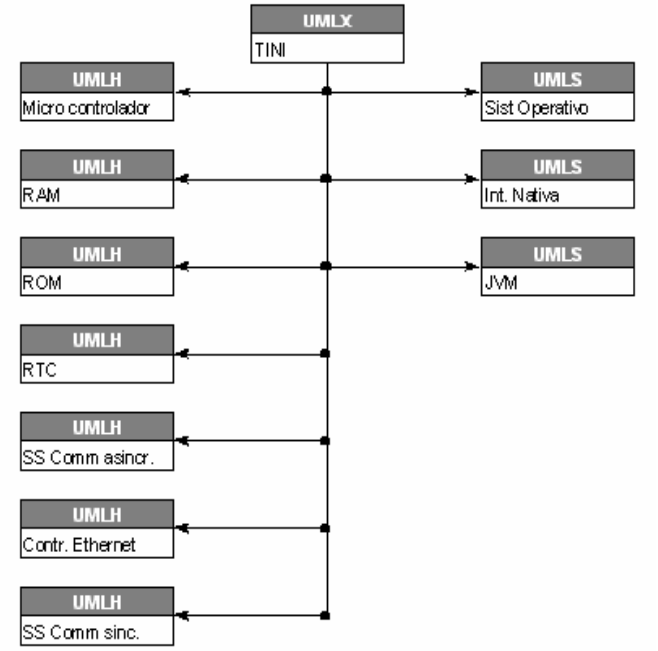

Figura 8: Jerarquía general de clases

Una descripción completa del sistema sería muy ilustrativa. Sin embargo, escapa al objetivo de este trabajo (los lectores interesados pueden consultar mayores detalles en www.lip.uns.edu.ar/tini y también en la referencia [11]). Por lo tanto, se profundiza el diseño en el sistema de comunicación serie que funciona bajo el estándar 1-Wire. Jerárquicamente, el sistema pertenece a la clase *Subsistema de Comunicación Asincrónica*. Esta clase engloba al hardware de comunicación RS-232 y 1-Wire. Esta conclusión se basa en que ambos puertos de comunicación están conectados a la misma UART. Por lo tanto, queda definida la jerarquía a partir de la clase común (Fig. 9)

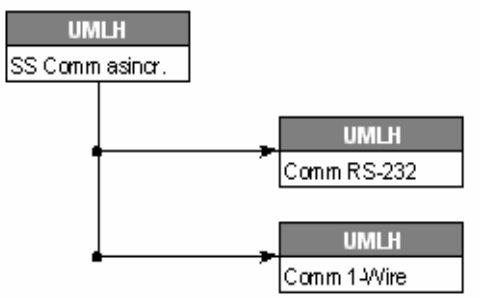

Figura 9: Jerarquía de SS Comm asincr.

En función de la estructura física, la interfaz 1-Wire (Fig. 11), se define a partir de la clase *Comm 1-Wire* (Fig. 12). Nótese que los componentes del diagrama UMLH se corresponden a los elementos físicos del circuito, mientras que las interfaces se corresponden a las líneas de conexión.

DIAGRAMA UML	
NOMBRE:	Sub-sistema de comunicación asincrónica.
COMPONENTES:	UART (integrada en el microcontrolador)
INTERFASES:	XRD0 TXD0 XRD1 TXD1 Canal de control. Canal de direcciones. Canal de datos.
RESPONSABILIDADES:	Comunica al micro controlador con dispositivos externos vía comunicación serie asincrónica.

Figura 10: Clase Subsistema de Comunicación Asincrónica.

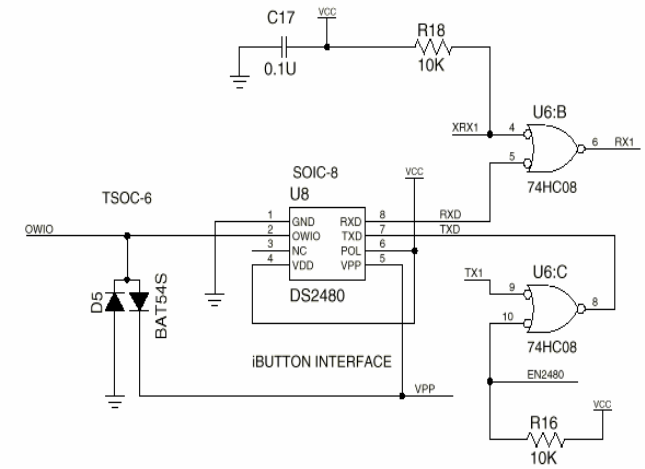

Figura 11: Esquema electrónico de la interfaz 1-Wire.

DIAGRAMA UML	
NOMBRE:	Comm 1-Wire
COMPONENTES:	U8 - DS2480. U6:B - 74HC08. U6:C - 74HC08. D5 - BAT54S. C17. R16, R18.
INTERFASES:	OWIO XRX1 TX1 RX1 EN2048
RESPONSABILIDADES:	Comunica a los dispositivos externos con el UART a través del estándar de comunicación serie asincrónico 1-wire.

Figura 12: Clase Comm 1-Wire.

En el campo del software, la comunicación 1-Wire es regulada por las clases contenidas en el paquete *com.dalsemi.onewire*. Este es un API (Application Program Interface) que se incluye en el ambiente de programación al momento de realizar la codificación.

DIAGRAMA UMLS	
NOMBRE:	com.dalsemi.onewire
ATRIBUTOS:	Ver definición provista por el fabricante
MÉTODOS:	public int reset() public abstract boolean getBit() public void putBit(Boolean BitValue) public void putByte (int byteValue) public int getByte()
RESPONSABILIDADES:	
Provee el código necesario para realizar las comunicaciones 1-Wire.	

Figura 13: Clase com.dalsemi.onewire.

Para mostrar cómo funciona el pasaje de UMLH a VHDL se analizará una compuerta que forma parte del sistema de comunicación 1-Wire. En particular se eligió el componente U6:B ubicado en el circuito integrado 74HC08. En términos funcionales, se puede decir que deja pasar la señal XRX1 cuando XRD está en nivel alto. La clase que modela al circuito se llama *Comparador XRX1*. Se muestra la clase que lo modela, la jerarquía de clases y la implementación en VHDL de la compuerta.

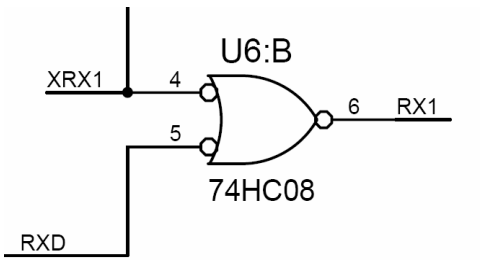

Figura 14: Comparador XRX1.

DIAGRAMA UMLH	
NOMBRE:	U6:B - 74HC08.
COMPONENTES:	U6:B - 74HC08.
INTERFASES:	XRX1 RX1 RXD
RESPONSABILIDADES:	
Habilita el paso de la señal XRX1 cuando RXD esta en nivel alto. RX1=(XRX1)AND(RXD)	

Figura 15: Clase U6:B - 74HC08.

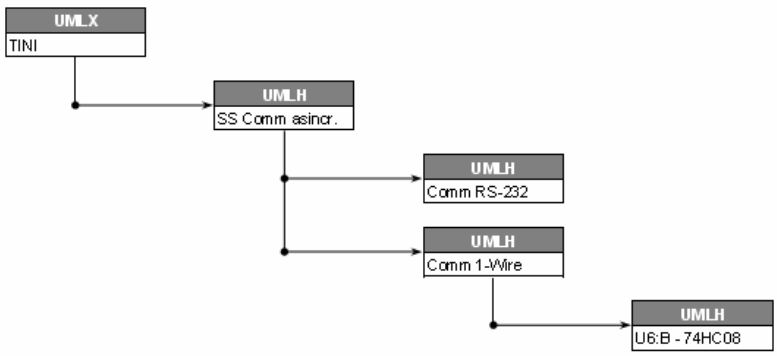

Figura 16: Jerarquía de clases hasta U6:B - 74HC08.

```
entity U6_B_74HC08 is
  port(
  XRX1 : IN std_logic;
  RXD : IN std_logic;
  RX1 : OUT std_logic;
  );
  end U6_B_74HC08;
```

Figura 17: Interfaz del componente U6_B_74HC08.

```
architecture Comportamiento of U6_B_74HC08 is
  begin
    RX1 <= XRX1 and RXD;
  end architecture U6_B_74HC08;
```

Figura 18: Arquitectura del componente U6_B_74HC08.

V. Agente de software de diseño de SI

A partir de una definición formal de los procesos involucrados en el diseño de SI se propone un agente de software para asistir en el diseño. La tarea del agente es optimizar al máximo posible los procedimientos a realizar. Con la ayuda de los diseñadores, el agente puede llevar adelante el proyecto. Gran parte de las tareas rutinarias pueden ser generadas por esta herramienta. Además, se puede generar un proceso de aprendizaje a partir de la interacción con los humanos. Es decir, el agente puede ir "recordando" las decisiones y aprender

El gran aporte del agente se hace presente en las tareas poco amigables para los diseñadores. En la Figura 19 se muestra algunas de estas tareas. Las tareas H e I constan en generar familias de clases implementables a partir de clases abstractas. El agente puede tener la inteligencia de hacerlo. En el caso de las tareas D y E se generan familias de objetos a partir de clases implementables dadas. Las tareas B y F evalúan el desempeño de los objetos en función a algún criterio (bajo consumo de energía, minimización de espacio u optimización de código) y se determina si la solución es apta o no. En el caso de C y G se comprueba el correcto funcionamiento de los objetos. Una manera de cumplir esa tarea es generar distintas situaciones de trabajo de los objetos y comprobar la correcta respuesta.

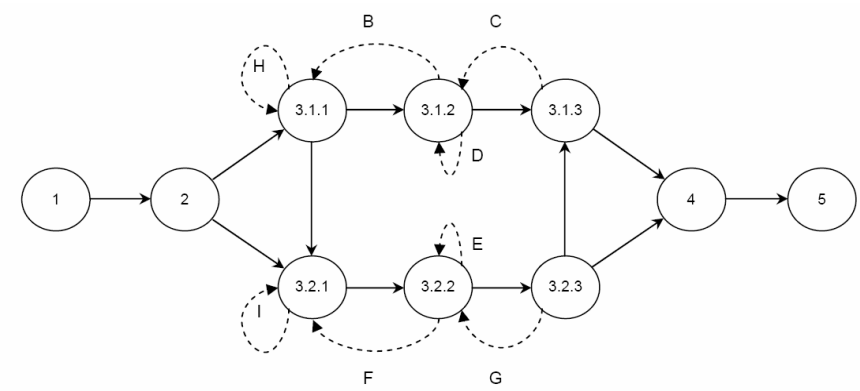

Figura 19: Tareas del agente.

VI Conclusiones y trabajo futuro

Se presentó un método de diseño para sistemas integrados. La estructura de clases facilita la resolución del problema por partes y regula los recursos a utilizar. Permite de manera natural el trabajo en equipo, lo cual facilita la implementación del sistema en un grupo de trabajo numeroso. Además permite ser implementado como un ambiente de trabajo en red. Gracias a la estructura de objetos se puede implementar una plataforma de trabajo sobre un servidor de bases de datos con varios usuarios trabajando simultáneamente.

La implementación resulta natural y es fácil de comprender debido al parentesco con las herramientas de la ingeniería en software. Se automatiza el diseño al extremo de generar automáticamente las sentencias de lenguajes de programación orientados a objetos y lenguajes de descripción de hardware. Minimizando los recursos a utilizar.

Finalmente, se propone un agente de diseño el cual ayuda a los integrantes del equipo a optimizar el diseño. Incluso se pueden lograr resultados novedosos tal como diseñar el SI a partir de minimizar espacio con ayuda del agente. La tubería de diseño es un esquema de trabajo lo suficientemente versátil como para solucionar las necesidades actuales de los diseñadores de SI.

Agradecimiento: Este trabajo fue parcialmente financiado por la SECYT-UNS.

Referencias

[1] J. Catsoulis, *Designing Embedded Hardware*, O'Reilly & Associates, 2002.

[2] C. Maxfield, *The Design Warrior's Guide to FPGAs*, Academic Press, 2004.

[3] A. Rushton, *VHDL for Logic Synthesis*, John Wiley & Sons, 1998.

[4] E. Sánchez Sinencio, A. Andreou, *Low-Voltage/Low-Power Integrated Circuits and Systems*, Wiley-IEEE Computer Society Pr, 1998.

[5] A. Burns, A. Wellings, *Real Time Systems and Programming Languages*, Addison Wesley, NY, 1997.

[6] M.Campione, *The Java(TM) Tutorial*, Addison-Wesley Pub Co, 2000.

[7] P. Dibble, *Real-Time Java Platform Programming*, Prentice Hall PTR, 2002.

[8] D. Loomis, *The TINI Specification and Developer's Guide*, Addison-Wesley Pub Co, 2001.

[9] M. Page-Jones, *Fundamentals of Object-Oriented Design in UML*, The Addison-Wesley Object Technology Series, 1999.

[10] J. Iparraguirre, C. Delrieux, *Método de Diseño de Sistemas Integrados Orientado a Objetos*, VI Workshop de Investigadores en Ciencias de la Computación, WICC 2004, Neuquén, Argentina, 2004.

[11] J. Iparraguirre, G. Ramoscelli y C. Delrieux, *Monitoreo y control de accesos mediante sistemas integrados.* RPIC, Reunión de Trabajo en Procesamiento de Información y Control. San Nicolás, Octubre de 2003, Páginas 729-735.