MÉTODO DE DISEÑO DE SISTEMAS INTEGRADOS ORIENTADO A OBJETOS

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

1. Introducción

Se considera un sistema integrado (SI) a aquel que contiene hardware y software dedicados a una tarea especifica. La concepción de estos diseños apunta a resolver problemas específicos, dejando en un segundo plano la versatilidad de aplicación. En cuestión de unos pocos años se produjo una gran revolución en esta área. Entre otros, el disparador de esta explosión tecnológica se debe a la telefonía celular. 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 que 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.

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.

2. Problemas Frecuentes

A continuación se mencionan los problemas más frecuentes en el proceso de diseño:

Mala 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.

Mala 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.

Falta de compatibilidad. La mayoría de los SI se comunican con otras plataformas de computación por medio de algún medio de comunicación. En muchos casos se desarrollan sistemas de comunicación de manera unilateral, impidiendo la comunicación con plataformas existentes en el mercado.

Imposibilidad de escalabilidad. Una alternativa a tener en cuente 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.

Imposibilidad de reutilización. 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.

Falta de 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 superior al esperado por los clientes. A pesar de cumplir con los requerimientos, un sistema no está correctamente diseñado si cuesta más de lo que los usuarios finales pueden pagar.

Excesivo tiempo de desarrollo. El tiempo al mercado debe minimizarse en todo proyecto de diseño de un SI. En la actualidad existe un gran número de personas que se encuentran desarrollando distintos sistemas con el fin de solucionar los mismos problemas.

3. Un método general

El método de diseño propuesto en este trabajo presenta un marco de trabajo que ordena el proceso de diseño durante toda su duración. 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.

A grandes rasgos, el método se puede dividir en tres pasos. El primer paso consiste en generar abstracciones en forma de *clases abstractas* basándose en los requerimientos. El segundo paso, consiste en subdividir a las clases primarias en *subclases implementables*. La tercera etapa consta en la creación de *objetos* los cuales pueden pertenecer al mundo del hardware o del software. Todo el proceso es regulado por un diagrama de *ciclo de vida* basado en el modelo V usual en la ingeniería de software (Fig. 1).

Todo el diseño se puede modelar con solo tres tipos de diagramas UML. El primero es el UMLX, *Abstract Unified Modelling Language*, el cual modela de manera abstracta al SI. El segundo es el HUML, *Hardware Unified*

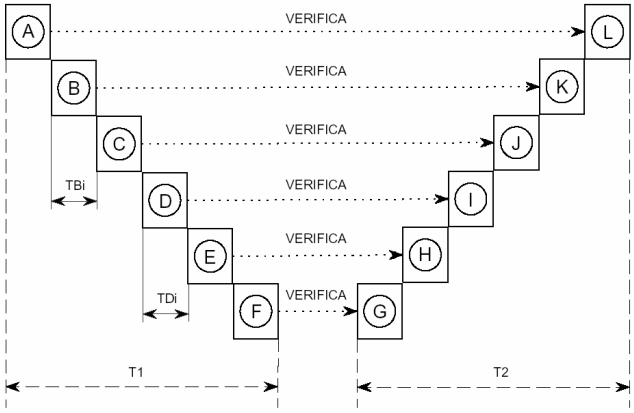

Figura 1: Ciclo de vida

Modelling Language, utilizado para modelar el hardware. El tercer diagrama es el UMLS, *Software Unified Modelling Language*, el cual modela al software.

La las clases generadas pueden heredar atributos de sus ancestros. Solo se deben respetar dos reglas. La primera de ellas es que clases del tipo UMLH heredan atributos de ancestros del mismo tipo o de UMLX. La segunda es que clases del tipo UMLH heredan atributos de ancestros del mismo tipo o de UMLX.

3.1 Ciclo de Vida

El ciclo de vida propuesto en este trabajo consta de 12 etapas. 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. En 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.

A. **Requerimientos generales.** En esta etapa se pacta con el cliente o usuario final los requerimientos del sistema a diseñar. Una vez que se definen los requerimientos, comienza formalmente el proceso. En este punto se definen las Clases Abstractas basadas en diagramas UMLX. En la Fig. 2 puede verse la estructura del diagrama.

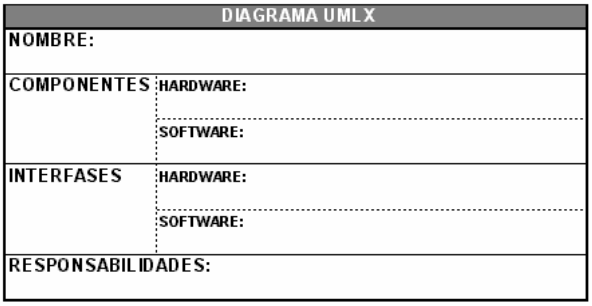

Figura 2: Diagrama UMLX

B. Requerimientos del hardware. En esta instancia se definen las tareas a cumplir por el hardware del dispositivo. En el diseño se deben tener en cuenta tanto las tareas funcionales como las condiciones del ambiente de trabajo. Los diagramas UMLH se completan en esta etapa. En la Fig. 3 se muestra la estructura del esquema.

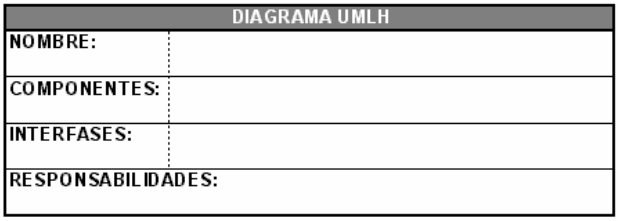

Figura 3: Diagrama UMLH

C. Requerimientos del Software. El software del dispositivo es definido en esta etapa. No se permite comenzar este modulo del diseño sin antes haber concluido el anterior. Unos de los requerimientos para el diseño del software es la definición del hardware.

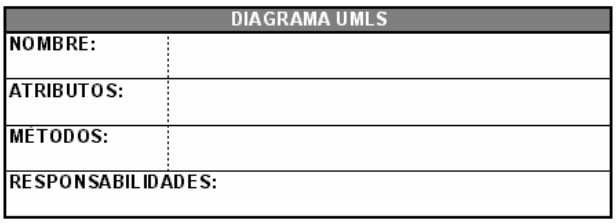

Figura 4: Diagrama UMLS

D. Análisis técnico - económico. Aunque a la vista de los diseñadores esta etapa sea innecesaria, no es recomendable que se omita su ejecución. Un diseño óptimo desde el punto de vista tecnológico, no siempre es viable a la hora de ser comercializado. En esta etapa se pueden realizar estudios característicos de otras áreas del conocimiento -análisis FODA, por ejemplo. En caso de que el diseño no resulte económicamente viable se debe volver a la instancia B.

E. Construcción del hardware. En este punto, los objetos del hardware se ensamblan en una unidad.

F. Construcción del software. El software se codifica en esta etapa.

G. Pruebas del software. En esta etapa se prueba el funcionamiento del software.

H. Pruebas del hardware. En esta etapa se prueba el hardware.

I. Verificación del costo de fabricación. El costo de fabricación del producto se define en esta etapa del proceso. Hasta este punto solo se puede estimar el mismo.

J. Integración y pruebas en el laboratorio. Una vez que se comprueba el correcto funcionamiento de las partes del sistema se debe probar el conjunto. Estas pruebas deben cumplir los requerimientos funcionales

K. Montaje y pruebas en el campo. Muchos de los SI funcionan en ambientes agresivos. Debido a esta condición particular, deben realizarse pruebas funcionales en el campo. Esta es la última oportunidad de detectar alguna falencia al diseño, por lo tanto, estas pruebas deben ser cuidadosamente planeadas.

L. Control de calidad, operación y mantenimiento. El diseño no concluye con la puesta en marcha del proyecto. A medida que transcurren las horas de funcionamiento, gran cantidad de datos pueden ser obtenidos del dispositivo. Las conclusiones se deben documentar con el fin de ser incluidas en nuevos emprendimientos.

4. Ejemplo de aplicación

Como ejemplo del uso de esta metodología, se mostrarán los principales pasos a seguir en el diseño de un servidor integrado. Se tomó como modelo la plataforma TINI (Tiny InterNet Interface) fabricada por Dallas Semiconductor. A grandes rasgos se puede decir que es una plataforma integrada en una placa (System on a Board) y un ambiente de operación Java. Toda la documentación al respecto puede encontrarse en http://www.ibutton.com/TINI/.

En primera instancia se creará la clase abstracta TINI, Fig. 5, 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 heredaran 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 5: Clase TINI

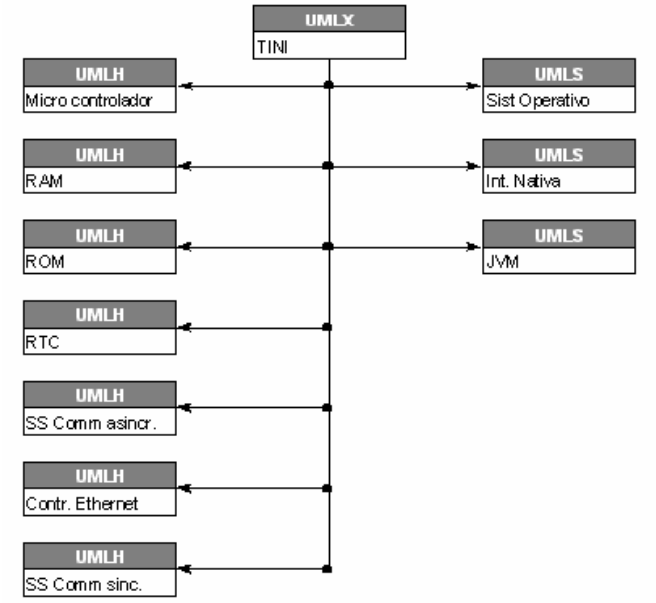

Figura 6: Jerarquía general de clases

En este ejemplo en particular, se profundizará en el diseño del sistema de comunicación serie que funciona bajo el estándar 1-Wire. A nivel hardware, el sistema pertenece a la clase *Subsistema de Comunicación Asincrónica*. Esta clase engloba al hardware de comunicación RS-232 y 1-Wire. La conclusión antes mencionada 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. 7)

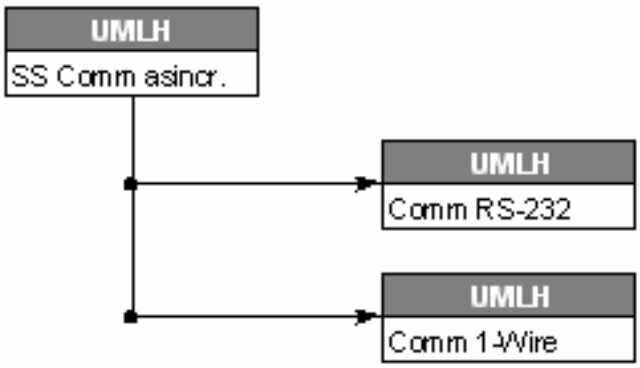

Figura 7: Jerarquía de la clase Subsistema de Comunicación Asincrónica.

En función de la estructura física de la interfaz 1-Wire (Fig. 9), se define la clase *Comm 1-Wire* (Fig. 10) Puede observarse 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 UMLH	
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 8: Clase Subsistema de Comunicación Asincrónica.

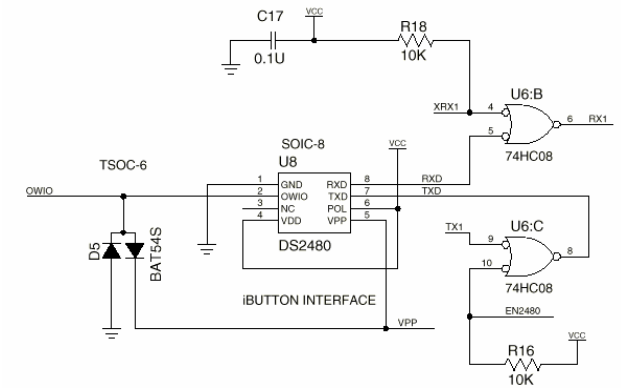

Figura 9: Esquema electrónico de la interfaz 1-Wire.

DIAGRAMA UMLH	
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 10: 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
METODOS:	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 11: Clase com.dalsemi.onewire.

5. Conclusiones

La característica más significativa de este método consta en la facilidad de aplicación y el orden implícito que contiene. 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.

Referencias

1. Campione, M: *The Java(TM) Tutorial*. Addison-Wesley Pub Co, 2000.

2. Dibble, P: *Real-Time Java Platform Programming*. Prentice Hall PTR, 2002.

3. Loomis, D: *The TINI Specification and Developer's Guide*. Addison-Wesley Pub Co, 2001.

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

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