Conceptos básicos en el desarrollo de sistemas Digitales de Seguridad

Cada vez existen más sistemas digitales en nuestro entorno. Muchos de ellos son sistemas críticos de seguridad, ya que la aparición de un fallo en su comportamiento puede producir peligros para la vida humana o para el entorno, así como pérdidas económicas. Este artículo presenta el trabajo que se está realizando para la obtención de un entorno integrado de desarrollo de sistemas digitales críticos de seguridad. Para conseguir una integración real, la información del sistema que se está desarrollando debe estar codificada en un modelo de comportamiento único, orientado a simulación, que se usará durante todo el ciclo de vida a lo largo de sucesivos prototipos.

Yolanda González Arechavala

Licenciada en Informática por la Universidad del País Vasco.
Es Investigadora en Formación del Instituto de Investigación Tecnológica de la ETS de Ingeniería de la UPCO. Este trabajo ha sido financiado en parte con una beca de la Asociación de Ingenieros del ICAI.

Fernando de Cuadra García

Dr. Ingeniero Industrial del ICAI. En la ETS de Ingeniería de la UPCO es Investigador del Instituto de Investigación Tecnológica, Profesor Propio Agregado y Jefe de Estudios de la titulación de Ingeniería Informática.

1. Introducción

Las normas [1, 2] y recomendaciones que se han de aplicar al desarrollo de los sistemas de seguridad hacen referencia a cómo se concibe el sistema y a cómo debe ser su desarrollo: concebido para ser seguro y desarrollado con una metodología de seguridad.

La línea de investigación que se describe en este artículo se basa en la experiencia adquirida en este tipo de proyectos así como en trabajos previos relativos a herramientas CASE y está enfocada a la obtención de un entorno integrado de desarrollo de sistemas digitales de seguridad. Para que la integración de todas las tareas de los desarrollos de este tipo sea un hecho, es necesario crear un modelo de comportamiento unificado que contenga toda la información necesaria para llevar a cabo dichas tareas. En los sistemas digitales de seguridad lo crítico es el comportamiento, por lo que los esfuerzos deben estar orientados a la comprobación de cómo se comporta el sistema. Esto se consigue mediante la utilización de un modelo orientado a simulación, que garantiza que el comportamiento está suficientemente descrito, y en ciertos casos sirve para la verificación de requisitos.

Existen otros proyectos en la actualidad que ponen de manifiesto la necesidad de estos entornos integrados [3-5].

Como primer paso, y con el fin de construir el modelo so-

bre unos conceptos básicos definidos sin ambigüedad, se hace un estudio sobre conceptos y nomenclatura utilizada. Para cada concepto básico, se propone una definición particular que se usará para los estudios posteriores.

A continuación, se presentan una serie de lenguajes de modelado usados por distintas metodologías para describir diversos aspectos de un sistema.

El siguiente paso es la presentación del modelo unificado de comportamiento, con sus requisitos y la explicación de lo que significa la unificación: integración de hardware y software, comportamiento reactivo y transformacional, estructura y comportamiento.

Finalmente, se presentan cuáles son los requisitos del entorno integrado propuesto así como las principales conclusiones obtenidas a lo largo del trabajo. En un artículo posterior se presentarán los resultados obtenidos por el simulador y del trabajo final.

2. Conceptos básicos

El estudio bibliográfico llevado a cabo como punto de partida de este trabajo revela que para algunos términos no existe una única definición, por lo que se ha decidido establecer un conjunto coherente de definiciones para construir el modelo unificado (sin pretender en ningún momento imponerlas como las únicas correctas).

El orden que se ha seguido es partir del concepto más global, como es el de sistema para llegar a caracterizar el sistema digital de forma unívoca a través del concepto de modelo, definiendo su estructura y su comportamiento. A continuación se presentan términos relativos al comportamiento, como son algoritmo, estado, evento y comunicación. El siguiente paso es presentar conceptos relacionados con aspectos temporales. Finalmente, se presentan algunos de los conceptos surgidos del paradigma orientado a objetos como son clase, instancia y herencia.

En este artículo se va a emplear metodología como se utiliza en ingeniería del software [6] y hace referencia a una colección consistente de tres elementos: un lenguaje de modelado (para capturar las propiedades del sistema), unos heurísticos de modelado (directivas informales de cómo usar el lenguaje de modelado) y un marco de organización y control de las tareas a realizar para el desarrollo de un sistema.

2.1. Sistema/modelo/sistema digital

2.1.1. Sistema

Todas las definiciones coinciden en que es un conjunto de elementos que persiguen un fin único, por lo que puede verse también como un único ente; tiene una interfaz determinada y perfectamente aislable. Las mejores definiciones son las presentadas en [7, 8] y la adoptada:

Definición: sistema es un conjunto de entes que se puede concebir como un único ente dentro de un determinado contexto, debido a una cohesión por comportamiento y estructura.

2.1.2. Modelo

Todas las definiciones [9, 10] están orientadas a la misma idea de abstracción de la realidad y descripción del sistema en su funcionalidad más importante, por tanto:

Definición: modelo es una representación abstracta de un sistema, que puede ser menos compleja y completa que éste y que describe su comportamiento y su estructura.

2.1.3. Modelo continuo-discreto (finito)

Las referencias estudiadas [11, 12] coinciden en la definición de estos términos. En muchos casos, los sistemas son mixtos: parte continua y parte discreta.

Definición: modelo continuo describe una evolución permanente en el tiempo mediante la utilización de ecuaciones diferenciales en su comportamiento y mediante variables continuas, con los papeles que desempeñan en su estructura (entrada, salida y variables de estado).

Definición: modelo discreto describe una evolución en

instantes determinados de tiempo, mediante la utilización de ecuaciones en diferencias, o eventos discretos significativos con las reglas de transición en su comportamiento, y mediante variables discretas, con los papeles que desempeñan en su estructura (entrada, salida y variables de estado).

Definición: modelo discreto finito es un modelo discreto con variables acotadas y, por tanto, con un conjunto finito de estados posibles.

2.1.4. Sistema digital

Para los sistemas digitales, el modelo que se elige es un modelo discreto finito [13], ya que el propio término digital implica la discretización y las variables de estado están acotadas por estar almacenadas en una máquina real.

Definición: sistema digital es aquél que se describe mediante un modelo discreto finito en sus variables y, por tanto, también discreto en el tiempo. Evoluciona a instantes de tiempo determinado y tiene sus variables acotadas.

2.1.5. Estructura/comportamiento interno/externo

Para modelar un sistema se debe definir su estructura y su comportamiento. Algunos autores no definen explícitamente ninguno de estos dos conceptos. Las citas que más se acercan a la definición adoptada son [6, 11] aunque discrepan en parte:

Definición: estructura representa aspectos estáticos del sistema, incluyendo la topología del mismo.

Definición: comportamiento describe aspectos dinámicos, como los cambios de estado, las variables de estado y posibles cambios dinámicos en la estructura.

La estructura y el comportamiento deben describirse siguiendo los dos puntos de vista de un sistema (externo-caja negra, interno-caja de cristal).

En el caso de la estructura, existe unanimidad en todas las definiciones encontradas, aunque muchos de los autores no hacen la distinción entre interna y externa. Son destacables las definiciones de [11]:

Definición: estructura externa de un sistema incluye las relaciones de entrada y salida del mismo.

Definición: estructura interna incluye conectores, almacenes de variables internas y referencia a los subsistemas que forman el sistema.

En el caso del comportamiento existe una mayor disparidad. En muchos casos ni se nombran los dos puntos de vista, y en los casos en los que se nombran [11] no llegan a hacer las distinciones que aquí se presentan.

Definición: comportamiento externo representa qué hace el sistema desde el punto de vista externo (cuando se usa para simulación, será un comportamiento equivalente al comportamiento interno, si es que está definido).

Definición: comportamiento interno describe los algoritmos de control que rigen los cambios de estado y de estructura interna.

2.2. Algoritmo, estado, evento

2.2.1. Algoritmo y Tiempo

El término "algoritmo" se utiliza en múltiples contextos y en todos ellos tiene un significado similar: es la descripción del comportamiento.

Definición: algoritmo describe el comportamiento (interno y externo) de un sistema digital. El algoritmo incluye información de tiempo, que puede ser tiempo cualitativo si sólo se especifica el orden de las transiciones y tiempo cuantitativo si lleva asociado una referencia de tiempo real.

2.2.2. Estado y variables de estado

El concepto de estado es universal:

Definición: estado de un modelo determinista de un sistema es toda la información tal que conociendo las entradas y el estado en un instante de tiempo, se determina de forma unívoca la salida y el nuevo estado del sistema.

El comportamiento de un sistema crítico de seguridad debe ser lo más determinista posible, para facilitar el control y la verificación exhaustiva del comportamiento. Cualquier indeterminismo debe detectarse lo antes posible.

A pesar de su importancia (memoria del sistema), las variables de estado son omitidas en muchos modelos de representación. Otros sí que las nombran [11]. La definición adoptada:

Definición: variables de estado son variables capaces de caracterizar de manera unívoca el estado de un sistema en un instante de tiempo determinado. Recogen la información relevante del pasado de manera que se pueda reproducir el futuro en cualquier instante (si el modelo es determinista).

2.2.3. Evento

En la literatura existen diversos conceptos relacionados con este término. Se habla de eventos secuenciales y eventos que modifican el flujo de control [14]. Para este estudio, la definición adoptada es:

Definición: evento es una causa que dispara o interrumpe un proceso de transición de estados. Es cualquier suceso ante el cual puede reaccionar un sistema.

2.3. Aspecto temporal del comportamiento

A continuación, se presentan algunos términos relativos al manejo del tiempo.

2.3.1. Comportamiento reactivo/transformacional

Existen dos tipos de comportamiento en los sistemas digitales [15, 16]: reactivo y transformacional. Ambos tipos de comportamiento suelen aparecer conjuntamente en los sistemas de seguridad.

Definición: comportamiento reactivo de un sistema se caracteriza por reaccionar continuamente a estímulos internos y externos, y por tanto, está dirigido por la ocurrencia de eventos. Relacionado con un comportamiento asíncrono.

> "El modelo orientado a simulación garantiza que el comportamiento está suficientemente descrito y en ciertos casos, sirve para la verificación de requisitos"

Definición: comportamiento transformacional de un sistema se caracteriza por tener predefinido el momento en que llegan las entradas y se produce la salida. Relacionado con un comportamiento síncrono.

2.3.2. Evento síncrono/asíncrono

Cuando el comportamiento de un sistema se ve afectado por la ocurrencia de un evento, se distingue entre síncrono y asíncrono [17] según sea el efecto que produce (punto de vista del receptor).

Un evento síncrono es aquél esperado o provocado por el propio sistema receptor, no interrumpe ninguna acción (momento predecible).

Un evento asíncrono es aquél que exige que el sistema receptor lo atienda en el instante que llega (momento impredecible)

2.3.3. Comunicación síncrona/asíncrona

Cuando se establece una interacción entre sistemas (mediante un mensaje o señal), se llama comunicación síncrona/asíncrona [14] según sea el comportamiento del sistema que inicia dicha comunicación (punto de vista del emisor).

Definición: una comunicación síncrona es aquélla en la que el emisor suspende sus tareas durante el tiempo que dura la comunicación.

Definición: una comunicación asíncrona es aquélla en la que el emisor no suspende sus tareas para esperar a que el receptor atienda a la comunicación.

Otro empleo del término síncrono en el mundo del hardware es el de un sistema que funciona provocando múltiples transiciones en paralelo según impulsos de un reloj real, frente a los sistemas asíncronos en los que las transiciones se producen mediante propagación sucesiva de cambios [9].

Este uso particular del término no nos parece relevante en este contexto.

2.4. Clases, instancias y herencia

La aparición hace ya unos años en la ingeniería del software del paradigma orientado a objetos, ha hecho necesario introducir algunas definiciones básicas. Existen muchos más conceptos relativos a este tema, pero se van a plasmar aquí sólo los más genéricos, aunque se han realizado estudios más amplios en esta línea. Las principales referencias utilizadas han sido [18-20].

En la definición de clase no suelen existir discrepancias:

Definición: clase (o sistema genérico) es la descripción genérica de una serie de atributos y operaciones sobre ellos, que puede ser particularizada al crear cada instancia (sistema).

Aunque al describir qué es una instancia de una clase no existen discrepancias, el problema surge con el término objeto, ya que algunas metodologías, como [14], utilizan dicho término con un enfoque similar al de clase, siendo en el resto utilizado como instancia de una clase. Por ello, los términos que se van a utilizar serán clase e instancia.

Definición: instancia es una entidad de programa creada a partir de una clase y que contiene datos y procedimientos que actúan sobre esos datos.

Aunque la definición de herencia que se va a presentar aquí no describe muy en detalle de qué se trata, sí que especifica qué tipo de mecanismo es:

Definición: herencia es un mecanismo de creación de nuevas clases a partir de una o varias ya existentes, de manera que la nueva clase hereda las propiedades de las clases padre.

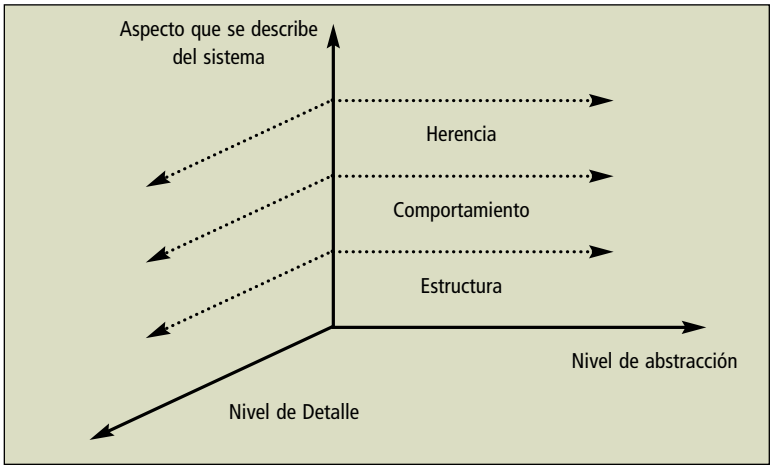

Figura 1: Caracterización de las descripciones de un sistema

3. Lenguaje del modelado

En la actualidad existe una gran cantidad de metodologías para el desarrollo de sistemas que pueden clasificarse según el paradigma de programación utilizado por cada una de ellas. Se van a presentar algunas de ellas (no siendo una revisión exhaustiva) con el fin de mostrar los lenguajes de modelado que utilizan.

3.1. Criterios de caracterización

Una descripción particular de un sistema se puede caracterizar situándola en un espacio de tres dimensiones (ejes ortogonales). Por un lado está el nivel de abstracción, que representa el grado de cercanía de los entes del modelo a los entes del producto final (software o hardware). Por otro lado, está el nivel de detalle, que es la descomposición progresiva de un problema en problemas más elementales (aplicable a requisitos, datos y sistemas en general). El tercer eje lo forma qué aspecto del sistema se describe en ese modelo: la estructura (datos, entradas y salidas con sus tipos), el comportamiento (variables de estado, relaciones causa-efecto) y en programación orientada a objetos, la herencia (relaciones padre-hijo entre clases o sistemas genéricos).

Gráficamente (Figura 1), existe la posibilidad de situar cada

descripción concreta de un sistema en una zona determinada de este espacio.

3.2. Estructurado

MCSE (Méthodologie de Conception des Systèmes Electroniques, [11]) es una metodología de programación estructurada, orientada a tiempo real y a sistemas empotrados que utiliza: modelo estructural (estructura de procesos o funciones, datos y comunicaciones, así como el tipo asociado a ellos), modelo de comportamiento (trata de describir los algoritmos en mayor o menor detalle y el tiempo asociado a ellos) y modelo ejecutivo (partiendo de los dos modelos anteriores, se realiza la separación entre qué va a software y qué a hardware).

3.3. Orientado al dato

JSD/JSP (Jackson Structured Design/Programming, [21]) está orientada al dato, ya que la estructura del sistema se determina por la estructura y evolución de los datos que se manejan. Existe una extensión para tiempo real. Los modelos propuestos son: diagrama de Jackson (descripción de las estructuras de datos y el comportamiento de las entidades), red de procesos (descripción funcional de los procesos y sus comunicaciones) y lenguaje JSP (descripción interna del comportamiento del proceso a un nivel de detalle mayor que se usa en la implementación).

3.4. Orientado a objetos

ROOM (Real-Time Object-Oriented Modeling, [6]) es una metodología orientada a objetos y a tiempo real. Describe dos conceptos de modelado de acuerdo a dos paradigmas:

- Paradigma de niveles de abstracción, según el nivel de detalle de los objetos en consideración: detalles de alto nivel y de bajo nivel.
- Paradigma de dimensiones de modelado, descomposición en tres dimensiones que representan los diferentes grados de libertad en el modelado: estructura (topología del sistema), comportamiento: (aspectos dinámicos) y herencia (incremento de la funcionalidad de cada subparte agregando nuevos comportamientos sobre lo que ya existe).

La combinación de ambos paradigmas es lo que dirige las actividades de modelado y se conoce como el Marco Conceptual (Conceptual Framework).

UML (Unified Modeling Language, [19]) es la unificación de las metodologías orientadas a objeto más utilizadas. Consiste en una serie de diagramas capaces de representar cualquier aspecto del sistema:

- Diagrama de Gestión del Modelo.
- Diagramas Estructurales Estáticos:
  - Diagrama de Clases, representa la estructura estática en términos de clases y relaciones.
  - Diagrama de Objetos, representa los objetos y sus relaciones.
- Diagrama de Casos de Uso, representa las funciones del sistema desde el punto de vista del usuario.
- Diagramas de Interacción:
  - Diagrama de Secuencia, es una representación temporal de los objetos y sus interacciones.
  - Diagrama de Colaboración, representación espacial de los objetos, enlaces e interacciones.
- Diagrama de Transición de Estados–Statecharts, representan el comportamiento de una clase en términos de estados.
- Diagrama de Actividades, describe el comportamiento de una operación en términos de acciones.
- Diagramas de Implementación:
  - Diagrama de Componentes, representa los componentes físicos de una aplicación.
  - Diagrama de Despliegue, representa la asignación de componentes a dispositivos físicos.

3.5. Hardware

El modelo general en el cual está basado el VHDL [22] es un lenguaje para describir hardware de sistemas digitales. Basado en tres modelos independientes: modelo de comportamiento (interpretación funcional de un sistema particular), modelo de tiempo (basado en el

paradigma estímulo-respuesta) y modelo de estructura (descomposición en subsistemas y sus comunicaciones).

4. Modelo unificado de comportamiento

El modelo unificado de comportamiento, presentado en este artículo como la pieza clave del entorno integrado, es capaz de modelar cualquier tipo de comportamiento digital relevante. Este modelo se llama MODUS que significa MOdelo Digital Unificado orientado a Simulación.

4.1. Principales requisitos de MODUS

Los requisitos impuestos a MODUS son:

- Sistemas Digitales: MODUS debe ser capaz de representar cualquier sistema digital. Por consideraciones de seguridad, puede restringirse algún tipo de comportamiento. Por simplicidad, se pueden reducir ciertos comportamientos a otros equivalentes ya definidos.
- Sistemas Empotrados de Tiempo Real. Los aspectos temporales deben estar incluidos en el modelo. Debe también considerarse la posibilidad de utilizar lógica temporal para la verificación formal.
- Orientado a simulación. La simulación es la mejor técnica para poder comprobar el efecto que tendría en el sistema de tiempo real cualquier decisión de diseño. Esto incluye tanto el diseño por prueba y error como los procedimientos sistemáticos de test.
- Modelado heterogéneo. Esto significa lograr la compatibilidad entre diferentes niveles de abstracción y detalle dentro del mismo prototipo.
- Unificado. MODUS representa los comportamientos transformacionales y los reactivos, la estructura, el comportamiento y la herencia, así como la influencia del hardware y del sistema operativo en el comportamiento del sistema que se está diseñando. Se utiliza tanto si el diseño se lleva a cabo siguiendo un paradigma orientado a objetos como estructurado.

La aportación principal de MODUS es la unificación que lleva a cabo. Por tanto, a continuación se presentan algunos de estos aspectos más en detalle:

4.1.1. Integración de la estructura y el comportamiento

Como se ha visto previamente, la estructura representa propiedades estáticas mientras que el comportamiento representa propiedades dinámicas así como cambios en la estructura.

Otro aspecto crucial del modelado, crítico en sistemas software, es la habilidad para representar la abstracción y la herencia. En software se permite la definición de sistemas genéricos o clases que puedan ser instanciadas para crear sistemas ejecutables así como la derivación de unas clases de otras clases.

Estos dos hechos plantean la pregunta de cómo se pueden modelar de manera precisa los cambios dinámicos de estructura (incluyendo la instanciación) y los mecanismos de herencia teniendo también en cuenta temas de seguridad como comportamiento determinista y previsible.

4.1.2. Integración de comportamiento reactivo y transformacional–orientación a objetos y estructurado

Los dos tipos de comportamiento coexisten en los sistemas de seguridad. El comportamiento reactivo se define a través de estados, eventos y acciones. El comportamiento transformacional se especifica normalmente con patrones clásicos estructurados, como secuencias, bucles y saltos condicionales.

El comportamiento reactivo está relacionado con el paradigma de programación orientada a objetos, que sugiere el uso de eventos y comunicaciones asíncronas. Los patrones transformacionales, por el contrario, se asimilan a la programación estructurada, en la que los eventos y las comunicaciones se realizan normalmente de manera síncrona.

En las aplicaciones de seguridad, uno de los requisitos más importantes es que la prueba sistemática (test) del comportamiento de un sistema pueda realizarse de la forma más completa posible. Desde este punto de vista, el modelo estructurado (sistemas transformacionales) es mejor que el orientado a objetos (sistemas reactivos) ya que es más fácil de probar [23, 24].

Es importante destacar que los patrones estructurados pueden ser implementados también en programación orientada a objetos. Lo importante es cómo se concibe el comportamiento, no el lenguaje elegido. El comportamiento estructurado, además de ser más sencillo de probar, obtiene diseños más seguros ya que es:

- Fácil de concebir y de entender.
- Fácil de representar, tanto con lenguajes formales como con gráficos.
- Está directamente controlado por la secuencia de las sentencias de código.

Los comportamientos reactivos y transformacionales pueden y deben coexistir en un único modelo a cualquier nivel de diseño, aunque en los sistemas de seguridad debe primarse el comportamiento transformacional (secuencial) por su facilidad de test. Los patrones orientados a objetos son compatibles con los patrones estructurados. Por tanto, MODUS integrará ambos tipos de comportamiento y ambos paradigmas.

4.1.3. Integración Hardware y Software

La idea es poder tener en cuenta las limitaciones que una plataforma hardware puede imponer a una aplicación software ejecutándose sobre ella (retrasos, desbordamiento de memoria, conflictos en el acceso a recursos). En este caso, el hardware y el software coexisten dentro del mismo sistema, pero a diferentes niveles de abstracción. El estado del hardware depende del comportamiento del software (por ejemplo, la petición de más memoria produce un cambio de estado en la memoria física), de la misma forma que el estado del hardware puede modificar el comportamiento ideal del software (retrasos en la ejecución de procesos, desbordamientos de memoria, recursos no disponibles).

> "Lo importante es cómo se diseña el comportamiento, no el lenguaje elegido"

Lo que se propone es caracterizar la plataforma hardware como un sistema separado con algunas variables de estado (estado de la memoria, estado de la CPU y disponibilidad de recursos). El siguiente paso será especificar el conjunto de reglas de "mapeo" que relacionarán el efecto que tienen las acciones software sobre el estado del hardware, así como la influencia del estado del hardware sobre el rendimiento del software (retrasos, excepciones o errores). Tanto el modelo de la plataforma hardware como las reglas de mapeo pueden reutilizarse para probar distintos modelos software sobre ellos. El modelo software se podrá probar sobre distintas plataformas hardware si están ya modeladas.

4.2. Formalismo visual para MODUS

Para cualquier gran desarrollo software se requiere una buena documentación, pero en el caso de los sistemas críticos de seguridad ésta es primordial. Los lenguajes gráficos son útiles para representar y entender diseños complejos. Es necesario utilizar distintos lenguajes para describir los diseños bajo diferentes puntos de vista (estructura, comportamiento e interacciones). Si los gráficos se crean de manera automática a partir del modelo unificado de la aplicación, todos ellos tendrán una estética similar y el trazado de conexiones será óptimo; además, se podrá obtener un ahorro de tiempo importante tanto al diseñar los diagramas como al actualizarlos.

Como se ha visto previamente, MODUS integrará el comportamiento reactivo (asíncrono) y el transformacional (síncrono). Para representar los comportamientos reactivos se ha elegido el formalismo visual Statecharts [15, 16] ampliamente utilizado por su potencia expresiva. Es una extensión de los diagramas de transición de estados (Figura 2), pero con mejoras significativas como son:

- Jerarquía de estado, uso de la descomposición de estados para estructurar comportamientos complejos.
- Ortogonalidad (concurrencia) mediante los estados AND.
- Generación y propagación de los eventos internos a niveles inferiores.

Para representar los comportamientos transformacionales se ha elegido el formalismo visual ANA [25] creado por Fernando

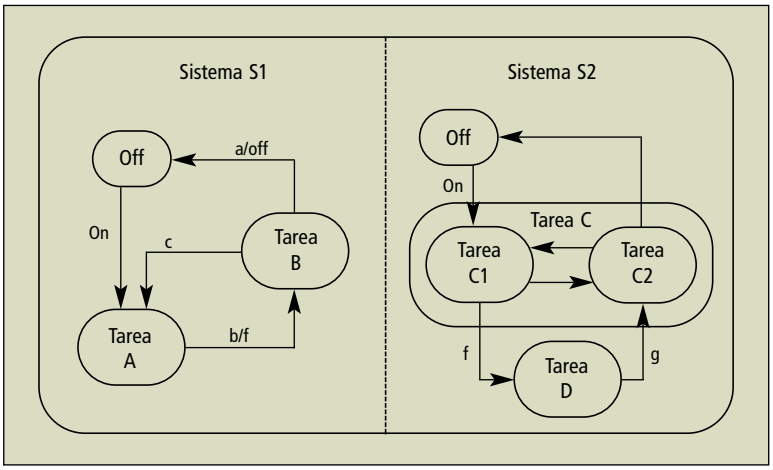

Figura 2: Comportamiento asíncrono mediante Statecharts

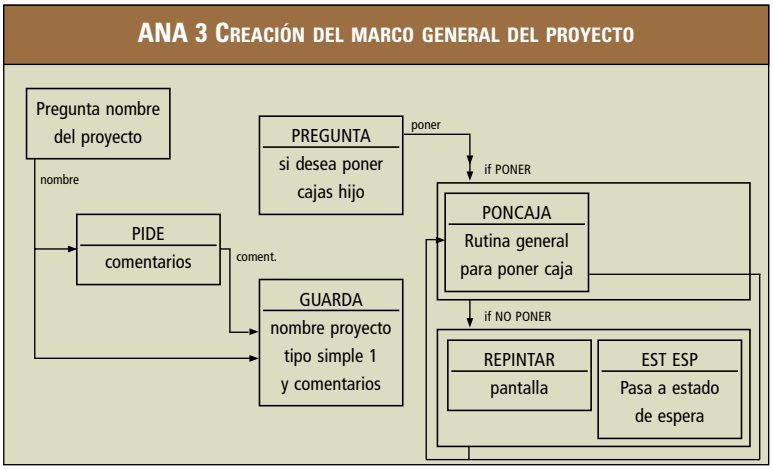

Figura 3: Comportamiento Síncrono mediante ANA

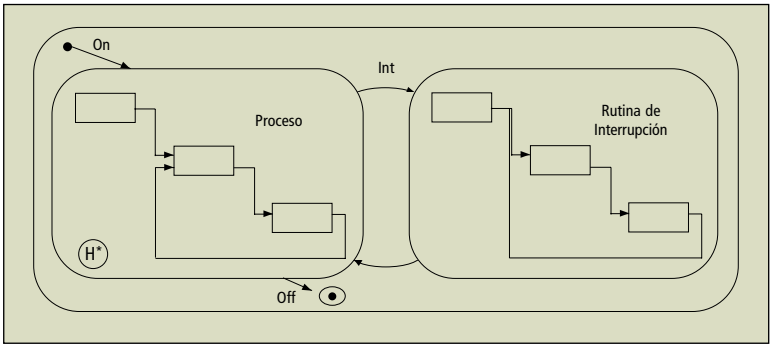

Figura 4: Diagrama ANA/Statechart

de Cuadra en el Instituto de Investigación Tecnológica (UPCo) y utilizado en diversos proyectos.

En ANA los patrones de comportamiento se reducen a secuencias, bucles, condicionales y sentencias de espera. La posición de las cajas representa explícitamente el control procedural, de manera que las flechas representan el flujo de datos. Así se consigue representar estructura y comportamiento en el mismo diagrama (Figura 3).

Además de otras representaciones gráficas adicionales (para cubrir la estructura y ciertos aspectos de relaciones) como DFD's, árboles o tablas, el formalismo visual que se utilizará para MODUS está basado en el formalismo Statecharts usando estados, transiciones, eventos y condiciones. El formalismo ANA es usado para modelar el comportamiento estructurado definiendo "superestados" con transiciones internas estructuradas (Figura 4).

4.3. Organización del proyecto

Los proyectos complejos requieren agrupar los sistemas en unidades de mayor orden, llamadas aplicaciones. Las aplicaciones son un conjunto de clases, siendo cada clase bien un sistema genérico o un tipo de datos genérico. Las aplicaciones pueden a su vez estar compuestas de otras aplicaciones. Un proyecto es la mayor unidad de gestión en MODUS, siendo la aplicación la unidad mínima de gestión. Cada aplicación puede estar etiquetada con un nivel de seguridad particular. La Figura 5 muestra esta organización gráficamente.

Ya que las clases pueden depender de otras clases usando distintos mecanismos (herencia, visi-

bilidad, posesión, uso), existirán relaciones de dependencia tanto entre clases como entre aplicaciones. En un buen diseño, las relaciones de dependencia más fuertes entre estas clases estarán siempre dentro de la misma aplicación, mientras que se podrán permitir dependencias débiles entre clases de distintas aplicaciones.

4.4. Modelo genérico de sistema

Los componentes básicos del comportamiento de un sistema genérico MODUS son:

- Variables de estado, relacionadas con los posibles estados en el caso de variables de estado de control.
- Estados, simples o compuestos (AND/OR/ANA).
- Transiciones, que comprenden eventos, estados, acciones y condiciones booleanas.
- Eventos, parte de las comunicaciones
- Expresiones, que son sentencias interpretables (simples o compuestas).
- Acciones, como creación/destrucción de sistemas, comunicaciones y asignaciones (simples o compuestas).

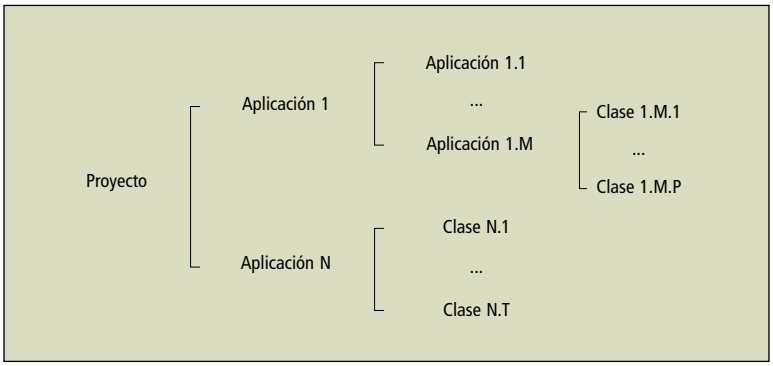

Figura 5: Organización MODUS

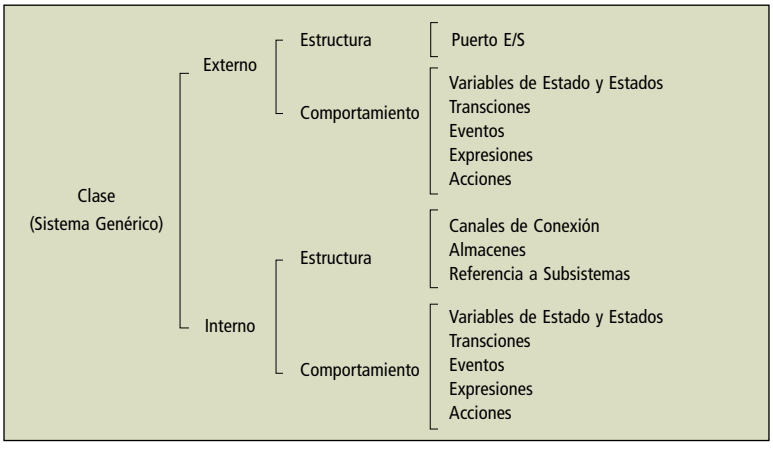

Figura 6: Información en MODUS

Los elementos básicos que se describen en la estructura de un sistema genérico MODUS son:

- Puertos de Entrada/Salida usados en la definición de la estructura externa. Un puerto se refiere a una forma particular de interacción.
- Canales de conexión, que se usan en la descripción interna, estableciendo caminos "físicos" de interacción entre subsistemas.
- Elementos de almacenaje que son parte de la estructura interna y almacenan los valores de la mayoría de las variables de estado.
- Referencias a subsistemas, ya que los subsistemas propiamente dichos están descritos aparte.

En el comportamiento y la estructura se utilizan también tipos de datos genéricos para definir las variables de estado auxiliares, elementos de almacenaje, datos que se intercambian a través de los puertos y en cualquier sitio donde sea necesario definir datos.

Con estos elementos (Figura 6), se construyen en MODUS clases básicas (sistemas genéricos), que pueden formar clases más complejas, aplicaciones y un proyecto. Estas clases son suficientes para el diseño a un nivel conceptual. Para el diseño físico, las clases básicas se irán especializando para incorporar las primitivas de estructura y comportamiento del lenguaje de programación elegido. Un entorno de diseño MODUS completo deberá contener bibliotecas para diferentes lenguajes de programación.

5. Entorno integrado para sistemas críticos de seguridad

La calidad de un producto software está directamente relacionada con la calidad de los ingenieros de desarrollo que trabajan en ella. El factor humano es el principal motivo del coste y la calidad de un desarrollo. Las metodologías de seguridad están muy restringidas por la documentación, revisiones, verificaciones y validación. Esto hace que los diseñadores tengan miedo de tomar decisiones, sean perezosos a la hora de cambiar un diseño aunque exista una solución mejor, y estén poco motivados y aburridos en su trabajo diario.

Es importante tener en cuenta este problema para especificar los requisitos del entorno integrado de desarrollo. Las metas generales son: reducir el coste de la aplicación, incrementar la calidad y mejorar la motivación de los ingenieros del software

El mejor camino para conseguir estas metas parece ser:

- Integrar todas las tareas que hay que realizar en el proceso de desarrollo software, usando un modelo único con toda la información para llevarlas a cabo.
- Máximo nivel de automatización, especialmente en las tareas rutinarias
- Diseño flexible mediante el uso de prueba y error, usando el simulador de MODUS para probar diferentes soluciones de diseño antes de seleccionar una. En los sistemas de seguridad, el uso de un simulador es muy útil para probar los diseños [26], y a veces es la única opción posible.
- Análisis de seguridad a cualquier nivel de detalle y de abstracción.
- Ciclo de vida basado en prototipos incrementales heterogéneos [3, 27]. Produce resultados parciales en etapas iniciales del proyecto y mejora en gran medida la creatividad. Los módulos críticos se desarrollarán primero, obteniéndose una importante experiencia en su desarrollo y detectando los cuellos de botella posibles.

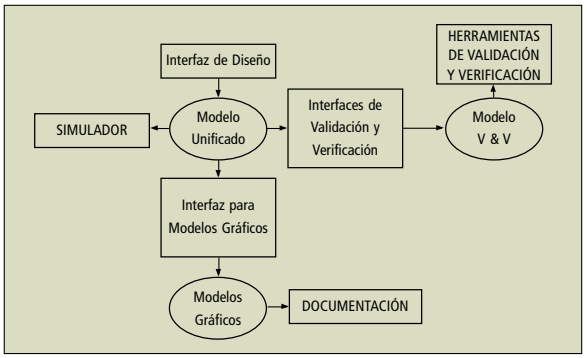

Figura 7: Entorno Integrado con MODUS

La funcionalidad del entorno integrado debe cubrir:

- Especificación de requisitos, usando métodos formales siempre que sea necesario.
- Simulación conjunta de diseños conceptuales y físicos.
- Generación automática de código para distintos lenguajes de programación.
- Validación y verificación, mediante métodos formales, casos de uso, escenarios y casos de test.
- Generación, ejecución (o simulación) y mantenimiento de casos de test.
- Análisis de seguridad a lo largo de diferentes fases y prototipos consecutivos.
- Herramientas de evaluación y gestión del proyecto.
- Generación automática de la documentación, incluyendo gráficos y texto.
- Configuración flexible de los métodos de trabajo, tanto en las áreas de trabajo formal como informales.

La pieza clave para conseguir el mejor entorno integrado será el modelo unificado MODUS, ya que será la principal fuente de información para todas las tareas que hay que realizar a lo largo del proceso de desarrollo. La tarea de diseño más creativa será precisamente la definición incremental del modelo.

La Figura 7 muestra la interacción del modelo unificado con algunas de las tareas que se llevarán a cabo en el entorno de desarrollo. Como se aprecia en la figura, existen una serie de modelos auxiliares, como los de verificación o gráficos, que serán generados de forma automática (siempre que sea posible) a través de interfaces especializados. Así se evita la información redundante.

6. Conclusiones

La conclusión principal del estudio es la necesidad de integración de las diferentes tareas en un entorno de desarrollo de

sistemas digitales de seguridad. Para conseguirlo la información del sistema se codificará en un modelo de comportamiento único que se usará durante todo el ciclo de vida a lo largo de sucesivos prototipos. Para poder tomar decisiones de diseño adecuadas, el modelo unificado debe estar orientado a simulación.

Durante el proyecto se realizarán varios prototipos, que irán avanzando en la misma medida en que el grupo de desarrollo adquiera un conocimiento más profundo del sistema a desarrollar. Muchas de las partes de un prototipo pueden ser utilizadas en el siguiente.

El éxito de un proyecto depende principalmente de sus recursos humanos. Por tanto, las herramientas y los procesos de desarrollo deben estar orientados al diseñador, permitiéndole comprobar distintas opciones mediante procesos de prueba y error.

La elección entre el paradigma estructurado o el orientado a objetos debe adaptarse al tipo de problema, aunque es posible que coexistan en el mismo proyecto obteniéndose así las ventajas características de cada uno de ellos. El paradigma orientado a objetos es mejor para el diseño de arquitecturas robustas y reutilizables, mientras que los patrones estructurados generan un comportamiento más seguro, fácil de probar (test) y fácil de entender.

Los diagramas deben ser una ayuda para facilitar el diseño, no algo impuesto que signifique un trabajo extra. En el caso de los aspectos dinámicos, se sugiere el uso de los Statecharts para representar el comportamiento reactivo y de los diagramas ANA para representar el comportamiento transformacional.

En la actualidad, los autores de este artículo están finalizando una consultoría sobre el desarrollo de sistemas críticos de seguridad para una compañía de señalización ferroviaria. Además, se están realizando dos tesis doctorales en este tema. Se ha finalizado la primera versión de MODUS además de la revisión del estado del arte. Se está diseñando el prototipo del simulador del modelo unificado. Las siguientes tareas que se tiene previsto realizar serán la automatización de los gráficos (para diseño, simulación y documentación) y los análisis de seguridad (uso de verificación formal y generación de casos de test).

Bibliografía

[1] "Draft Standard IEC 61508: Functional safety of electrical / electronics / programmable electronic safety-related systems", International Electrotechnical Commission, Geneva, 1998.
[2] "Draft prEN 50128: Railway Application Software for railway control and protection systems," CENELEC: European Committee for Electrotechnical Standarization, Brussels, November 1995.
[3] "GUARDS: Generic Upgradable Architecture for Real-Time Dependable Systems" ESPRIT 20716.
[4] "COMITY: COdesign Method and Integrated Tools for Advanced Embedded SYstems" ESPRIT 23 015.
[5] M. Weber, "Combining Statecharts and Z for the Design of Safety-Critical Control Systems" Lecture Notes in Computer Science, vol. 1051, pp. 307-326, 1996.
[6] B. Selic, G. Gullekson, and P. T. Ward, Real-Time Object-Oriented Modeling, Wiley, 1994.
[7] M. Broy, "(Inter-)Action Refinement. The Easy Way" in Codesign: Computer-Aided Software/Hardware Engineering, I. Press, Ed. Piscataway, New Jersey: IEEE Press, pp. 67-97, 1995.
[8] R. S. Pressman, Ingeniería del Software: Un enfoque práctico, Third Ed.: McGraw-Hill / Interamérica de España, S.A., 1995.
[9] R. H. Thayer and M. C. Thayer, "Glossary of Software Engineering Terms" in Software Engineering: A European Perspective, I. C. S. P. Tutorial, IEEE Press, pp. 587-647, 1993.
[10] J. Banks and J. S. Carson, Discrete-Event System Simulation, Prentice-Hall, 1984.
[11] J. P. Calvez, Embedded Real-Time Systems, Wiley, 1993.
[12] W. Y. Fletcher, Engineering Approach to Digital Design, Prentice-Hall, 1980.
[13] L. Lamport, "A simple approach to specifying concurrent systems" Communication ACM, vol. 32(1), pp. 32-45, 1989.
[14] S. Shlaer and S. J. Mellor, Object Lifecycles: Modeling the World in State, Prentice Hall, 1992.
[15] D. Harel, "STATECHARTS: A visual formalism for complex systems" Science of Computer Programming, vol. 8, pp. 231-274, 1987.
[16] D. Harel and M. Politi, Modeling Reactive Systems with Statecharts, Computing McGraw-Hill, 1998.
[17] P. A. Laplante, Real-Time Systems Design and Analysis: an engineer's handbook. Los Alamitos, California: IEEE Computer Society Press, 1992.
[18] A. Morales Lozano and F. J. Segovia Pérez, Programación Orientada a Objetos: Aplicaciones con Smalltalk, Paraninfo, 1993.
[19] G. Booch, J. Rumbaugh, and I. Jacobson, The Unified Modeling Language: User Guide, Addison-Wesley, 1998.
[20] P. Müller, "Introduction to Object-Oriented Programming using C++" Globewide Network Academy (GNA) November 18, 1996.
[21] M. Jackson, System Development, Prentice-Hall, 1983.
[22] R. Lipsett, C. Schaefer, and C. Ussery, VHDL: Hardware Description and Design, Second Edition Ed: Kluwer Academic Publishers, 1990.
[23] R. V. Binder, "Trends in Testing OO Software" IEEE Computer, pp. 68-69, 1995.
[24] M. Falla, Advances in Safety Critical Systems: Results and Achievements from the DTI/EPSRC R&D Programme in Safety Critical System, DTI/EPSRC, 1997.
[25] F. de Cuadra, "El Problema General de la Optimización de Diseño por Ordenador: Aplicación de Técnicas de Ingeniería del Conocimiento" Tesis Doctoral de la Universidad Pontificia Comillas, 1990.
[26] N. Storey and J. F. Craine, "Environmental Simulation in the Development of Safety-Critical Software" 3rd. International Conference on Reliability, Quality Control and Risk Assessments, Washington DC, 1994.
[27] J. M. Sanz Engel, "IPTES: Tecnología de prototipado incremental para sistemas de tiempo real empotrados" Comunicaciones de Telefónica I+D, vol5, nº 1, pp. 43-59, 1994.