Modelado de sistemas supervisores distribuidos y abiertos usando redes de Petri y orientación a objetos

Gladys Bastidas Gustin - Vicerrectora de investigaciones
Institución Universitaria Tecnológica de Comfacauca
gbastidas@unicomfacauca.edu.co
E. Villani, F.Junqueira, P. E. Miyagi
Escola Politécnica, Universidade de São Paulo, São Paulo, Brasil
(evillani, fabri, pemiyagi) @usp.br

Resumen

El modelado de sistemas supervisores para ambientes automatizados puede ser visto como una tarea que involucra técnicas y métodos de dos áreas principales: ingeniería de control e ingeniería de software. En este contexto, el propósito de este trabajo es introducir un nuevo abordaje para el modelado de sistemas supervisores distribuidos y abiertos, basado en la unión de técnicas tradicionales de ingeniería de software (como la orientación a objetos) y con modelos formales de los sistemas dinámicos a eventos discretos (como las redes de Petri). En el primer nivel de abstracción, es usado el modelo de referencia de procesamiento distribuido y abierto (RM-ODP) como marco arquitectural estándar para la construcción de un sistema distribuido y abierto. Basado en el modelo de referencia ODP, los diagramas en el lenguaje de modelado unificado UML son construidos como un segundo nivel de abstracción. Finalmente, las redes de Petri (el tercer nivel de abstracción) son usadas a través del proceso con la finalidad de garantizar la coherencia entre los modelos en UML desde el análisis de requerimientos hasta la implementación de manera a proveer modelos formales del sistema.

Palabras clave: Sistema supervisor, ODP, UML, Petri nets.

I. Introducción

En el contexto de sistemas de automatización, el mejoramiento operacional y funcional de los sistemas supervisores ha sido un tópico constante de investigaciones. Las principales características de un sistema supervisor son las siguientes: es un sistema de tiempo real que debe reaccionar concurrentemente a comandos aperiódicos que pueden provenir de los usuarios y/o de los sistemas de alto nivel. Estos sistemas deben continuamente monitorear cambios de estado en los sistemas de control local y en planta [1]. Entre las responsabilidades de un sistema supervisor están el monitoreo y la detección de situaciones anormales, la escogencia entre estrategias de control, integrar y realizar el intercambio de información entre sistemas de control local, etc. La complejidad de un sistema supervisor puede variar en un amplio rango. Particularmente, este trabajo se enfoca en los casos donde esta complejidad justifica una implementación distribuida y el uso de lenguajes de programación orientados a objetos.

En automatización de sistemas, el desarrollo de sistemas de control distribuido y abierto puede proveer soluciones para problemas relacionados con aspectos tales como flexibilidad, preservación de la inversión, conectividad, interoperabilidad con otros sistemas y portabilidad.

Respecto a sistemas supervisores, estas ventajas son particularmente deseadas una vez que el sistema supervisor es responsable por la integración de un número de sistemas de control local y equipamiento.

Considerando estas características, el modelado de sistemas supervisores puede ser visto como una tarea que envuelve técnicas y métodos de dos áreas principales: ingeniería de software y de control. Respecto a la ingeniería de software, la principal particularidad del modelado de sistemas supervisores es que el software de supervisión debe ser verificado y validado considerando el comportamiento del objeto controlado (planta más sistemas de control local), lo cual significa que un modelo del comportamiento de este objeto debería ser construido. Desde el punto de vista de la ingeniería de control, el modelado de sistemas supervisores puede ser clasificado como un problema del dominio de los Sistemas Dinámicos a Eventos Discretos (DEDS), una vez que el sistema supervisor, el sistema de control local y la planta pueden ser modelados por estados discretos y eventos instantáneos.

En este contexto, el propósito de este trabajo es introducir un nuevo abordaje para el modelado de sistemas supervisores distribuidos y abiertos basados en la utilización de dos técnicas tradicionales de ingeniería de software (tales como los conceptos de orientación a objetos) con modelos formales de DEDS (tales como redes de Petri). En un primer nivel de abstracción, el Modelo de referencia de Procesamiento Distribuido y Abierto (RM-ODP) es usado como un marco arquitectural estándar para la construcción de sistemas distribuidos y abiertos. Basado en el RM-ODP, los diagramas del lenguaje de modelado unificado (UML) son construidos como segundo nivel de abstracción. Finalmente, las redes de Petri (el tercer nivel de abstracción) son usadas a través del proceso con la finalidad de garantizar la coherencia entre los modelos en UML del análisis de requerimientos a la implementación y suministrar modelos formales del sistema.

Este artículo está organizado como sigue: sección 2 discute las principales ventajas del uso combinado de RM-ODP, UML y Petri nets. Sección 3 ilustra el abordaje propuesto, usando como un ejemplo el modelado de un sistema contra incendios en un edificio inteligente. Sección 4 presenta los estudios de caso usados para desarrollar y probar el abordaje propuesto y finalmente, la sección 5 introduce algunas conclusiones y trabajos futuros.

II. RM-ODP, UML y PETRI NET: una asociación sinérgica

El RM-ODP (modelo de referencia de procesamiento distribuido y abierto) es un esfuerzo conjunto de la Organización Internacional de Estandarización (ISO) y la Unión Internacional de Telecomunicaciones (ITU) para proveer un marco para el desarrollo de sistemas distribuidos y abiertos heterogéneos, de gran escala en un ambiente [2]. Este define conceptos y reglas estructurales para especificar sistemas distribuidos y abiertos de acuerdo a diferentes puntos de vista. Cada punto de vista de la ODP provee una abstracción del sistema enfocándose en una particular área de interés ([3] y [4]). Los cinco puntos de vista del ODP son:

1. *Punto de vista empresa:* se enfoca en el propósito, la finalidad, políticas y restricciones de un sistema. Los principales conceptos del punto de vista de la empresa son: roles, comunidades y objetivos.

2. *Punto de vista información:* se enfoca en la información y en el procesamiento de la información. Aquí se trata cómo la información es estructurada, cómo ésta cambia, los flujos de información, y las divisiones lógicas entre funciones independientes dentro del sistema.

3. *Punto de vista computacional:* se enfoca en una descomposición funcional del sistema en objetos, los cuales interactuan por sus interfaces.

4. *Punto de vista ingeniería:* trata sobre cómo la interacción entre objetos del sistema es soportada (infraestructura requerida para soportar el procesamiento distribuido).

5. *Punto de vista tecnología:* se concentra en el hardware individual y en los componentes de software, los cuales componen el sistema (tecnologías adecuadas para soportar el procesamiento distribuido).

Es importante observar que el RM-ODP no define la forma de representación de cada punto de vista, aunque impone un modelaje orientado a objetos. Esa cuestión es resuelta por el uso de los diagramas de UML [5].

En el contexto del modelaje orientado a objetos, el UML ha sido adoptado más y más como un patrón. Entre sus ventajas está la habilidad de representar un sistema con diferentes visiones y niveles de detalle. Brevemente, el UML puede ser descrito como un lenguaje que, comenzando con un grupo de elementos básicos (tales como clases, casos de uso, interacciones, estados, etc.) y las posibles relaciones entre estos elementos (tales como composición y herencia entre otras), define un grupo de diagramas para representar diferentes visiones del mismo sistema.

A pesar de sus ventajas, el UML tiene aún algunos puntos a ser mejorados ([6] y [7]). Entre los problemas está la descripción del comportamiento del sistema y cuestiones relacionadas con paralelismo, sincronismo y concurrencia entre procesos. Con la finalidad de mejorar la descripción del comportamiento de aspectos discretos, algunos trabajos fueron desarrollados relacionando UML y Redes de Petri ([7] y [8]).

Entre los formalismos para representar DEDS, las Redes de Petri se caracterizan por su habilidad de manejar operaciones de secuencia, paralelismo, conflicto y exclusión mutua en sistemas. Estas características hacen de ella una herramienta promisoria para describir y analizar sistemas concurrentes y de tiempo real. Adicional a su poder de modelado, las redes de Petri se constituyen en una descripción gráfica y en un formalismo matemático.

En este contexto, este trabajo propone un abordaje para el modelado de sistemas a gran escala. Particularmente, el foco de este artículo es el modelado de sistemas supervisores, esto es, cómo obtener un modelo en redes de Petri del sistema supervisor teniendo como punto de partida una definición general de los propósitos del sistema. Seguidamente, los modelos deben ser analizados con la finalidad de verificar si este cumple los requerimientos definidos. Finalmente este debería ser implementado al traducirlo del modelo en redes de Petri a un software que use un lenguaje de programación orientada a objetos. El análisis e implementación no están dentro del alcance de este artículo, sin embargo algunas cuestiones claves son discutidas y algunas referencias son presentadas en la sección 5.

III. El abordaje propuesto

De acuerdo al abordaje propuesto, el modelado de sistemas está dividido en dos fases principales: en la primera fase el sistema supervisor es una "caja negra". Los modelos producidos en esta fase representan las interacciones del sistema supervisor con los sistemas de control local y los subsistemas de la planta, los cuales están también considerados como "cajas negras". En la segunda fase, los modelos del sistema supervisor, los sistemas de control local y los subsistemas de planta son detallados.

Fig. 1 presenta las principales actividades del abordaje y modelado y su relación con las fases generales de desarrollo de software y puntos de vista del ODP. Estas actividades son descritas a seguir utilizando como ejemplo el sistema contra incendios de un edificio inteligente. Debido a limitaciones de espacio, sólo modelos parciales son presentados y muchas simplificaciones son consideradas.

III1 *1ra fase: Sistema Supervisor como caja negra*

En la primera fase son definidas las actividades de la empresa, las comunidades y los roles, fijando de esta manera los límites del sistema. Después el control local y los subsistemas de la planta son identificados basados en la estructura del sistema a ser supervisado. Simultáneamente, los diagramas de casos de uso del UML son construidos considerando los roles y los actores y mostrando las relaciones entre ellos y el sistema (sistema supervisor + sistema de control local + planta). Cada caso de uso es entonces detallado en un diagrama de colaboración del UML, indicando la relación entre el sistema supervisor, la planta y los subsistemas de control local (previamente identificados).

El sistema contra incendios, tomado como ejemplo, tiene como su principal propósito detectar fuego y actuar en los sistemas del edificio con el fin de minimizar los potenciales daños del fuego.

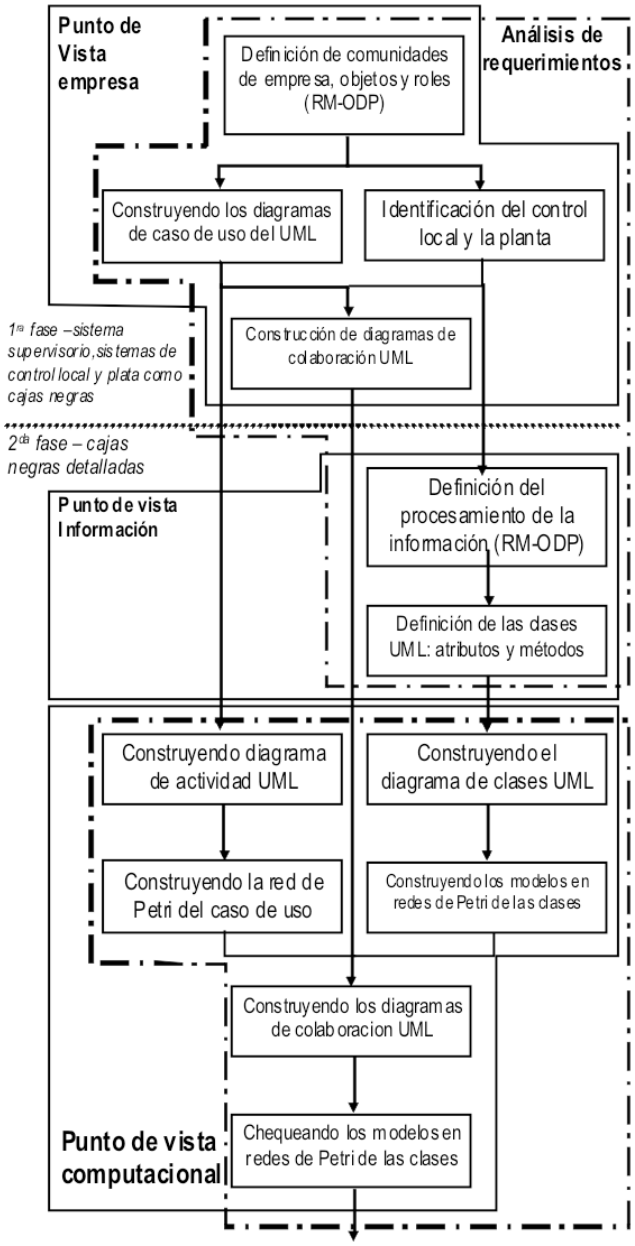

Figura 1. Abordaje de modelado.

Algunas de las actividades del sistema contra incendios son:

- Detectar fuego por el monitoreo de un grupo de sensores (sensor de temperatura, sensor de fuego, sensor de humo, etc.)
- En caso de incendio actuar sobre los sprinklers, insufladores, etc.
- En caso de incendio, comunicarse con otros sistemas del edificio como el sistema de aire acondicionado, ventilación y calentamiento – sistema HVAC, el sistema de iluminación, el sistema de control de acceso, etc., esto con el fin de que estos sistemas puedan realizar sus estrategias de incendio y así ayudar al sistema de incendio (ejemplo: el sistema HVAC ayuda a remover el humo y prevenir que el humo se esparza a través del edificio).
- En caso de incendio, comunicar de éste a los bomberos.
- Proveer rutinas y funciones para la prueba de los equipamientos y estrategias de incendio.

Considerando estas funciones un ejemplo de un diagrama de casos de uso del UML es presentado en figura 2.

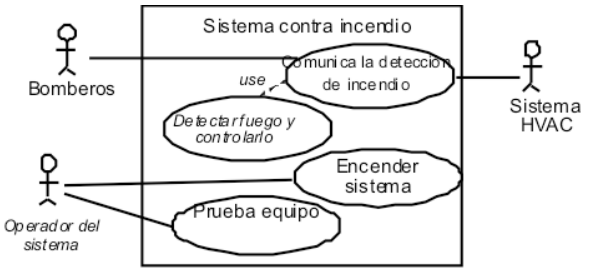

Figura 2. Diagrama de casos de uso del UML.

Entre los componentes de la planta y de los subsistemas de control local están el sistema de sprinkler, el sistema de insuflamiento, el sistema de sensores, etc. Un ejemplo del diagrama de colaboración para 'detectar fuego y controlarlo' es presentado en la figura 3.

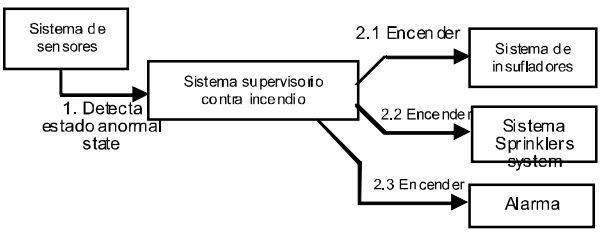

Figura 3. Diagrama de colaboración del UML.

III2 2da fase: La caja negra detallada

En la segunda fase, los dos principales aspectos del modelado de sistemas están sincrónicamente enfocado a: objetos y flujos de actividades. Para el modelado de los objetos, el sistema supervisor, el sistema de control local y la planta son detallados definiendo cual es la información que necesita ser modelada y como ésta es procesada. Basado en esta

descripción, las clases del UML para el sistema supervisor, el sistema de control local y la planta son definidos.

En el ejemplo algunas de las informaciones del sistema supervisor son:

- Estatus de los sprinklers, insufladores y otros equipamientos.
- Salidas de los sensores.
- Información desde y hacia los otros sistemas del edificio: sistema HVAC y estatus de los elevadores, número de personas en las zonas con fuego, rutas de escape que deben ser señalizadas por el sistema de iluminación, etc.

Un ejemplo del procesamiento de la información es:

- Si la salida del sensor de temperatura no es normal y la salida de sensor de humo es humo detectado, una alerta debe ser enviada a los otros sistemas del edificio y el estatus de los sprinklers e insufladores en la zona deben ser modificados (ellos deben ser encendidos).

Ejemplos de las clases del UML son presentados en la figura 4. En esta figura hay ejemplos de clases del modelos de la planta (bombas, sensores, sprinklers, tanque de reserva); del modelo de control local (controlador de la bomba) y del sistema supervisor (detector de fuego y controlador de incendio).

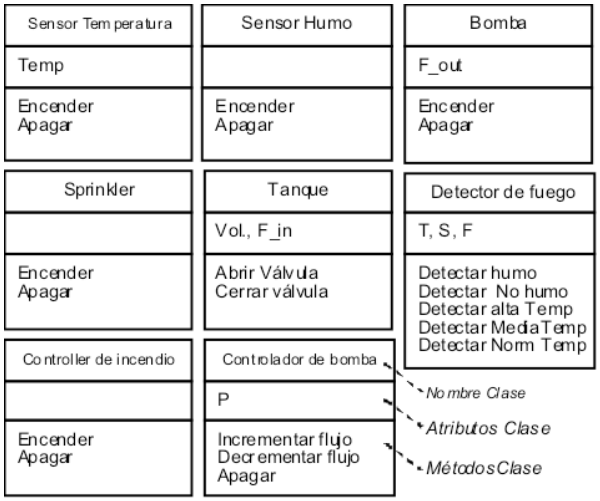

Figura 4. Clases UML

El diagrama de clases para estas clases es presentado en la figura 5.

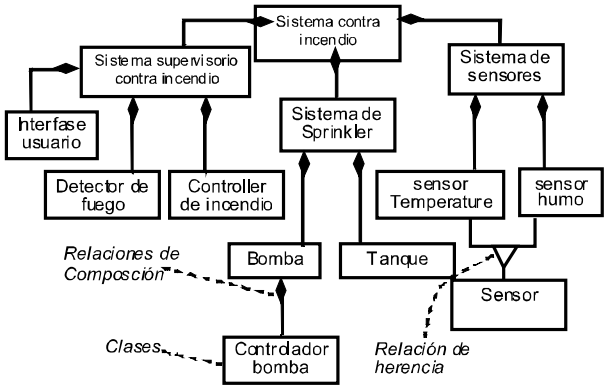

Figura 5. Diagrama de clases UML.

El próximo paso es construir un modelo en redes de Petri de cada una de las clases del UML. Para el modelado de las clases, las siguientes declaraciones son consideradas:

- Una red de Petri de capacidad unitaria modela el comportamiento de la clase.
- Los atributos de una clase son modelados como el grupo de atributos de las marcas de las redes de Petri.
- El primer atributo de una marca en una clase es la identidad del objeto.
- Un objeto es representado por una marca en la red de la clase, o por un grupo de marcas con la misma identidad.
- Una llamada a un método es representada por un disparo de la transición.

El tipo de red de Petri usada para el modelado de cada clase no es definida por el abordaje. Este puede variar de acuerdo a las características de cada clase y de cada sistema. La única restricción es que las interfaces entre los objetos deberían ser modeladas por el disparo de transiciones en todas las clases, de acuerdo a lo que es definido en [8]. Una clase simple sin atributos puede ser modelada por una red de Petri Condición/Evento (C/E) [9]. Un ejemplo es una clase sensor de humo (Fig. 6). Las transiciones representadas como una barra son transiciones internas. Las transiciones representadas como un rectángulo negro son métodos de otras clases que son llamados por la clase sensor de humo, el nombre del dueño del método está en itálico. Finalmente, las transiciones representadas por un rectángulo blanco son

métodos provistos por la clase que pueden ser llamados por las clases de otros sistemas.

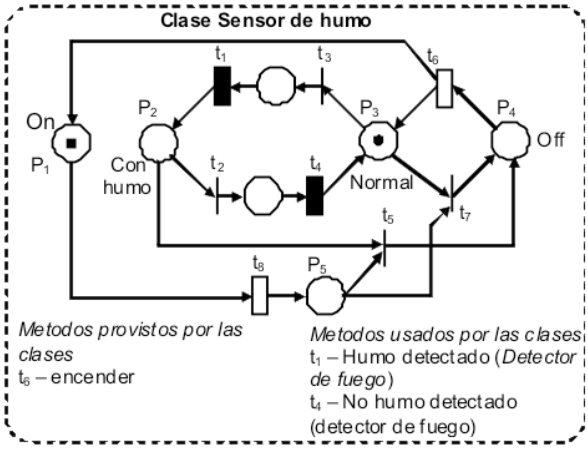

Figura 6. Ejemplo de clases de modelado con redes de Petri C/E.

Otro ejemplo de clase modelado por una red de Petri C/E es la clase controlador de incendio (figura 7) del sistema supervisor. Una vez que el fuego ha sido detectado en el edificio, este objeto es responsable por tomar las acciones apropiadas.

Una clase con solo atributos discretos puede ser modelada por una red de Petri coloreada [10], una red Predicado/transición [11], E-MFG [12] o cualquier otra red de Petri de alto nivel. Un ejemplo de una clase modelada por una red Predicado/ transición es la clase detector de fuego, presentada en la Fig. 8. En este caso, las marcas tienen atributos. Una marca en P_1 tiene el atributo S, indicando el estatus del sensor de humo (si S=1, entonces el humo es detectado en el edificio, si S=0 entonces no se ha detectado humo). El valor de S es definido por la acción asociada con las transiciones t_1 (A: S=1) y t_0 (A: S=0), que son métodos llamados por el sensor de humo. Similarmente, el atributo T en P_2 indica el estatus de la temperatura (T=0 –temperatura normal, T=1 –temperatura media, y T=2 –temperatura alta). Los atributos F de la marca en P_4 indican el estatus del controlador de incendio (F=0 – *Controlador de incendio en stand by*, F=1 - *Controlador de incendio controlando incendio*). Las condiciones asociadas con t_6 (C: S=1 y T1 y =0), y t_7 (C: S=0 y T=0 y F=1) detectan fuego o la ausencia de fuego respectivamente y, de acuerdo a esto, enciende o apaga el controlador de incendio (t_8 y t_9).

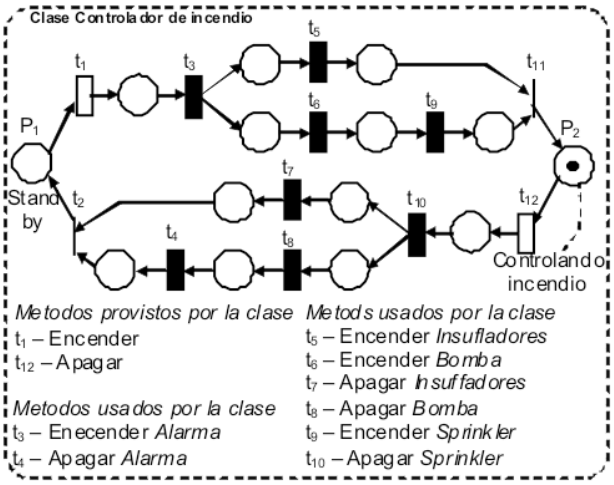

Figura 7. Ejemplo de clase modelada con redes de Petri C/E

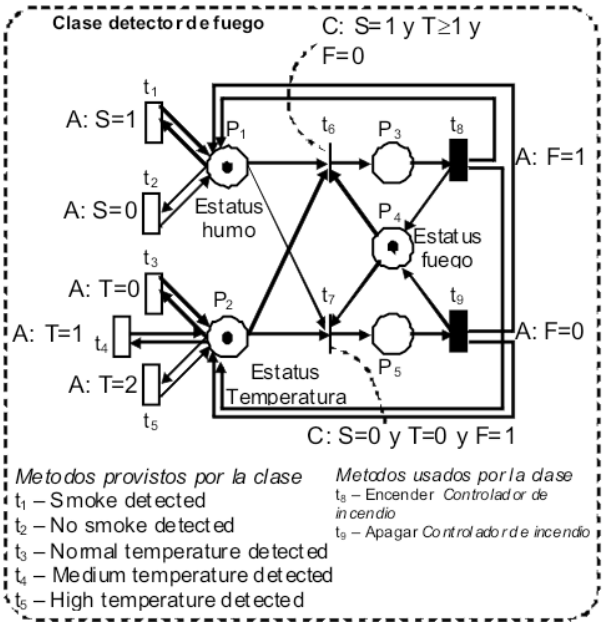

Figura 8. Ejemplo de clase modelada con red de Petri predicado/ transición.

Finalmente, una clase con variables continuas y dinámica continua puede ser modelada por una red de Petri Predicado/Transición Diferencial. Ejemplos pueden ser encontrados en [13].

Para el modelado del flujo de actividades, cada caso de uso es detallado en un diagrama de actividad del UML que muestra las actividades realizadas por el sistema para ese caso de uso. Este modelo es entonces traducido a un modelo en redes de Petri. figura 9 para el caso de uso 'Detectar fuego y controlarlo' de la figura 2. El correspondiente modelo en red de Petri es obtenido de asociar una

transición de la red de Petri a cada actividad del diagrama de actividad del UML.

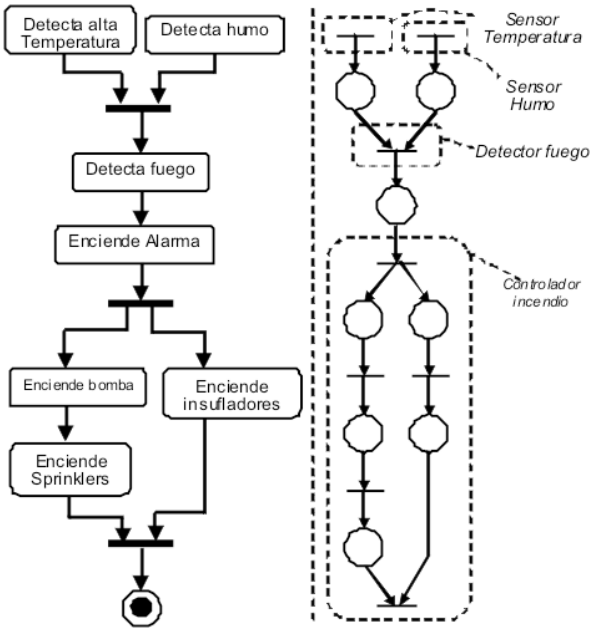

Figura 9. Diagrama de actividad UML y correspondiente modelo de caso de uso en red de Petri.

El próximo paso es determinar quien realizará cada actividad del diagrama de actividades del UML. En este ejemplo, el sensor de temperatura debe 'detectar temperatura alta', el sensor de humo debe 'detectar humo' y así en adelante.

Una vez esto está hecho, el diagrama de colaboración del UML es construido (figura 10). Cada vez que una actividad del diagrama de actividad del UML no es realizada por el mismo objeto que realice la actividad previa, debe ser una interacción correspondiente entre las dos clases en el diagrama de colaboración del UML. Por ejemplo, la actividad 'detectar humo' es realizada por el sensor de humo y la próxima actividad ('detectar fuego') es realizada por el detector de fuego, esto significa que el sensor de humo debe comunicar la detección de humo al detector de fuego, esta comunicación es representada por la interacción 1.2 del diagrama de colaboración del UML. Además de estas interacciones el diagrama de colaboración del UML debería también contener interacciones para cada actividad que corresponda a una llamada de método, tal como 'encender la alarma', 'encender la bomba', etc.

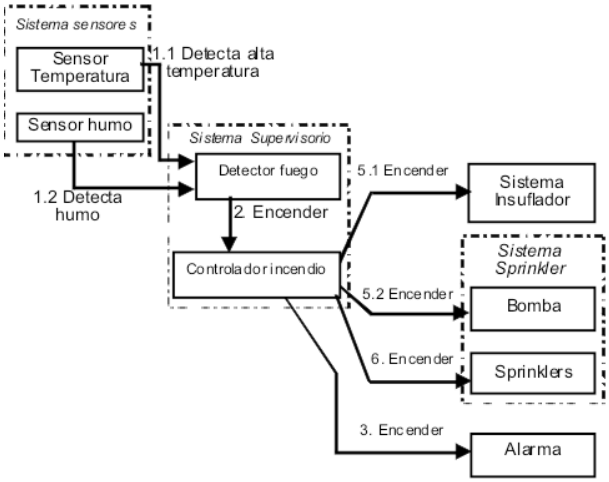

Figura 10. Diagrama de Colaboración UML.

El último paso de la figura 1 es 'chequeando los modelos de las clases en redes de Petri'. Básicamente, cada transición de la red de Petri modela un caso de uso y cada interacción del diagrama de colaboración del UML debería corresponder a una transición del modelo de clase. Ejemplo: para el caso de uso de la figura 9, la transición de la actividad 'detectar humo' es la transición t_3 del sensor de humo. La interacción 1.2 del diagrama de colaboración del UML de la Fig. 10 es el par de transiciones t_1-sensor de humo/t_1-detector de fuego.

Los pasos de la Fig. 1 pueden ser subsecuentemente repetidos en un abordaje *top-down* o *bottom-up*. El modelo de un objeto puede ser descompuesto/compuesto o detallado por ejemplo por una relación de herencia.

IV. Estudio de casos

La metodología propuesta ha sido aplicada a dos casos de estudio principales: el sistema de gerenciamiento y supervisión en edificios inteligentes (estudio de caso 1) y el sistema supervisor para un ingenio azucarero (estudio de caso 2). El objetivo de estos estudios de caso es probar el abordaje propuesto para sistemas reales con alto grado de complejidad.

Estudio de caso 1

Edificios inteligentes pueden ser definidos como los edificios donde la integración de sistemas y nuevas tecnologías son usadas con el fin de maximizar la productividad de sus ocupantes,

permitir un eficiente gerenciamiento de recursos y minimizar costos [14], [15]. La integración de los sistemas de edificio (sistema HVAC, sistema de iluminación, sistema de seguridad, etc.) es hecha a través de un sistema de gerenciamiento que recibe datos de los sistemas, y los transmite a otros sistemas del edificio e interfiere en la evolución de los sistemas del edificio con el fin de mejorar el desempeño del edificio. En este caso de estudio es considerado el modelado del sistema de gerenciamiento, el sistema de seguridad, el sistema HVAC, el sistema contra incendios y el sistema de iluminación de un hospital. Los modelos presentados en la sección 3 son versiones simplificadas de los modelos para el sistema contra incendios.

Estudio de Caso 2
Un ingenio azucarero consiste de una serie de procesos mecánicos y químicos agrupados por fases. Algunas fases podrían ser clasificadas como procesos por lotes, mientras que otras son esencialmente continuas. Este estudio de caso presenta el modelado del sistema supervisor de un ingenio azucarero. Particularmente este es determinado por las cuestiones relacionadas a cómo la dinámica continua puede ser incorporada a el abordaje con el objetivo de hacer posible su aplicación a sistemas supervisores híbridos.

V. Conclusión

Este trabajo propone un abordaje para el modelado de sistemas supervisores distribuidos y abiertos. El punto de innovación de este abordaje es la asociación sinérgica de RM-ODP, UML y redes de Petri. Los puntos de vista del RM-ODP guían el proceso de modelado. Usando el UML los aspectos de orientación a objetos son destacados a través de los diferentes diagramas, mientras una representación formal del sistema es suministrada por el uso de redes de Petri.

Respecto al análisis e implementación, los modelos generados pueden ser analizados por dos diferentes estrategias: verificación formal o simulación. La verificación formal de propiedades ha sido estudiada como la descomposición del sistema en objetos, que pueden ser explorados con el fin de descomponer un problema de análisis complejo, que concierne a un modelo global de un sistema en un grupo de problemas simples que conciernen sólo uno o pequeños grupos de objetos.

La simulación, ha sido explotada en cómo la simulación, además de ser una herramienta de análisis, puede también ser un paso intermedio entre el modelado de un sistema supervisor y su implementación. En este sentido, una plataforma para simular modelos en redes de Petri distribuidas y abiertas está siendo desarrollada. Antes de ser simulado en un ambiente distribuido, algunas cuestiones relacionadas a la implementación del sistema deberían ser abordadas, antes como la organización de los objetos en componentes (Diagrama de componentes del UML) y los cuales son los nodos del sistema (diagrama de utilización). Estos aspectos son parte del punto de vista ingeniería del ODP. Aquí es importante resaltar que una de las ventajas de la simulación distribuida es el manejo fácil de modelos a gran escala.

Una vez las redes de Petri han sido simuladas y los errores corregidos, el modelo en redes de Petri de los nodos del sistema supervisor puede ser gradualmente remplazado por software y conectado usando el mismo esquema. Esta es otra ventaja del abordaje propuesto. Así, errores en la traducción de redes de Petri a un lenguaje de programación orientado a objetos puede también ser detectado.

Durante esta fase de codificación, las cuestiones relacionadas al punto de vista tecnología deben ser abordadas, tal como la escogencia de un lenguaje de programación.

VI. Bibliografía

[1] R. Champagnat, et al. "Modelling and Simulation of a Hybrid System through Pr/Tr PN-DAE Model", in *Proceedings of ADMP'98 3rd International Conference on Automation of Mixed Processes, 1998* Reims.

[2] ISO/IEC, Open Distributed Processing -Reference Model Part 3: architecture, *International Standard, 1995*

[3] Faroqui, K., Logrippo and L. Meer, J., " The ISO reference model for open distributes processing: an introduction", *Comp. Nets. and ISDN Systems* 27, 1995, 1215-1229.

[4] Gaspoz, J. P., "Methodology for the development of distributed telecommunications service", *Journal Systems Software* **33, 1996,**: 253-271.

[5] Booch, G., Rumbaugh, J. and Jacobson, I., *The Unified Modelling Language User Guide*, Addison-Wesley Longman, Inc. Harlow, England, 1998.

[6] Giese, H., Graf, J. and Wirtz, G. (1999), "Closing the gap between object-oriented modelling of structure and behaviour", in *Proceedings of the Second International Conference on the Unified Modeling Language*, Colorado, USA, 1999.

[7] L. Baresi, M. Pezze, "On Formalizing UML with High-Level Petri Nets", *Concurrent Object-Oriented Programming and Petri Nets – Lecture Notes in Computer Science*, Springer-Verlag.

[8] M. Paludetto. *Sur la commande des procédés industriels: une méthodologie basée objects et réseaux de Petri*, Thèse de Doctorat, Université Paul Sabatier, Toulouse, France, 1991.

[9] T. Murata, "Petri Nets: Properties, Analysis and Applications" *Proceedings of the IEEE*, 77, 4, April, 1989, 541-580.

[10] K. Jensen, *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1, Basic Concepts*, Monographs in Theoretical Computer Science, Springer-Verlag, 1997.

[11] W. Reisig, G. Rozenberg (Eds.), *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, Lecture Notes in Computer Science*, vol. 1491, Springer-Verlag, 1998.

[12] Miyagi, P. E. and Santos, D. "Enhance Mark Flow Graph to control autonomous guided vehicles", in *Proceedings of the CAPE:857-865, 1995,* Chapman Hall, New York, 1995.

[13] Miyagi, P. et al., "Design of Hybrid Supervisory System using UML and Petri nets", in *Proceedings of the 8th IEEE Conference on Emerging Technologies and Factory Automation*, Antibes Juan-les-pins 2001.

[14] Becker, R. (1995), "What is an intelligent building?", in *Proceedings of the Intelligent Buildings Congress*, Tel-Aviv 1995, 229-240.

[15] Clark, G., Mehta, P and Prowse, R., 1995, "Intelligent integrated building management systems", *in Proceeding of the Intelligent Building Congress,* Tel-Aviv, 1995, 9-18.