REPRESENTACIÓN DE OBJETOS OASIS EN TEORÍAS Y SU TRADUCCIÓN A UN MODELO DE CONCURRENCIA

Pedro Sánchez Palma, Patricio Letelier Torres e Isidro Ramos Salavert
Departamento de Sistemas de Información y Computación
Universidad Politécnica de Valencia
e-mail :{ppalma | letelier | iramos} @dsic.upv.es

RESUMEN

La validación temprana de requisitos del software durante el desarrollo de un sistema de información es un aspecto de reconocida importancia. La animación automática a partir de la especificación de requisitos del sistema puede llegar a ser una valiosa ayuda en dicha tarea. OASIS es un lenguaje formal para especificación de modelos conceptuales. En este trabajo se presenta una formalización del proceso de obtención de un prototipo dada una especificación OASIS, un modelo básico de ejecución y un Modelo de Concurrencia como son las Redes de Petri clásicas. Si definimos las características formales que debe cumplir un prototipo para que satisfaga una especificación OASIS habremos conseguido una representación ejecutable fiel al modelo original. El planteamiento que se hace del trabajo es un comienzo que permite ser extendido al estudio de otros tipos de redes (Object Petri Nets, Coloured Petri Nets, etc.) u otros modelos de concurrencia.

1. INTRODUCCIÓN

OASIS [13][14] es un lenguaje formal para la especificación conceptual de sistemas de información. Actualmente existe un creciente interés en incorporar total o parcialmente técnicas formales para modelar sistemas de información [16][3]. Otros lenguajes de especificación formales con enfoques similares al ofrecido por OASIS son ALBERT [1][2], CMSL [22] y TROLL [4]. La propuesta metodológica basada en OASIS es OO-Method [16]. Se ha desarrollado una herramienta CASE basada en OO-Method [15][17].

Una funcionalidad atractiva es la posibilidad de animar la especificación del sistema mediante un prototipo generado automáticamente. En esta línea, hemos experimentado usando Redes de Petri [19] y lenguajes de Programación Lógica Concurrente [11][12] como dominio semántico para especificaciones OASIS. El extender los conceptos expuestos a otros modelos de concurrencia puede hacerse redefiniento el concepto de estado y adecuándolo al planteamiento escogido. El objetivo de este trabajo es presentar una justificación formal del proceso de traducción y de la obtención del prototipo de especificaciones OASIS en un entorno de programación concurrente. En nuestro caso, la problemática de verificar la corrección de la animación respecto del modelo conceptual se ve apoyada por dos factores importantes: el modelo conceptual se especifica en un lenguaje formal y el modelo de ejecución está inspirado en la semántica operacional de dicho lenguaje formal [10]. Esto, por sí mismo determina la corrección exigida y provee un camino de formalización, objetivo de este trabajo.

El trabajo se estructura comenzando con una presentación formal de los conceptos objetuales, tales como, clase, objeto, estado, mundo, etc. Posteriormente, estos conceptos se relacionan con la semántica asociada a una especificación OASIS determinada en el Modelo de Ejecución descrito en [9][10]. En último lugar, se formalizan las condiciones necesarias para que la implementación de un prototipo utilizando Redes de Petri sea consistente con el modelo planteado.

2. DEFINICIÓN DE CLASE Y OBJETO

La mayoría de los conceptos que se exponen a continuación ya han sido definidos en otros trabajos como [21][23] pero los consideramos necesarios para sentar las bases de las definiciones posteriores.

Definición 1.1 Un **Esquema de Clase EC** es una cuatro-tupla (A, E, C, P), siendo **A** el alfabeto de Atributos, **E** el alfabeto de Eventos, **C** un conjunto de axiomas (fórmulas en lógica dinámica) y **P** un conjunto de ecuaciones en un Álgebra de Procesos.

Sea *SORT* un conjunto de nombres de géneros (*sorts*). Para cada atributo $a \in A$ de un esquema de clase *EC*, existe un sort $S_{a \in A} \in SORT$. Denotamos por $|S_a|$ al conjunto soporte[1] del sort S_a.

Definición 1.2 Una **Clase** en Oasis puede representarse como la tripleta (name, EC, ID), siendo **name** el nombre de la clase, **ID** los mecanismos de identificación de sus instancias (objetos) y **EC** un esquema de clase o plantilla de los aspectos estáticos y dinámicos que comparten todas las instancias.

Definición 1.3 Un **Objeto** es un proceso observable [18] cuya evolución viene caracterizada por la ocurrencia de eventos y cuyos aspectos estáticos y dinámicos están determinados por el EC de la clase a la que pertenece.

Los objetos pueden verse desde una doble perspectiva [14]: *estática* y *dinámica*. Desde un punto de vista estático poseen un estado representado (entre otros) por valuaciones de sus atributos en el sort apropiado; por otro lado (perspectiva dinámica), la ocurrencia de eventos causa una transición entre estados. El lenguaje de la lógica dinámica utiliza dos lenguajes: un conjunto *F* de fórmulas de primer orden como abstracción de las propiedades que caracterizan el estado, y un conjunto *P* de procesos como abstracción de cambios de estado.

3. SEMÁNTICA DECLARATIVA

La semántica declarativa de la lógica dinámica viene dada [21] mediante estructuras de Kripke $S = (W, \pi, m)$ tales que W es un conjunto cuyos elementos se llaman mundos posibles; π es una función que asigna a cada fórmula (perteneciente a F) un subconjunto de W en los que es cierta (π: $F \rightarrow 2^W$), y **m** una función con dominio del sort de procesos P y con codominio un conjunto de relaciones binarias sobre W (asignando a cada elemento de P una relación de accesibilidad entre mundos, es decir,

$$m: \qquad P \rightarrow (W \rightarrow W)).$$

Definición 2.1 Una **Estructura S** es una dos-tupla (W, π) con W un conjunto y π: $F \rightarrow 2^W$ una función.

Definición 2.2 Para cada estructura S = (W, π), se define la función prop : $W \rightarrow ((A \times S_A) \cup (E \times S_A^n))$ como prop(w) = {($_{a \in A}${a} × {|sort(a)|) ∪ ($_{e \in E}${e} × {|sort^n(e)|))}tal que (π(F) = w y de manera que podemos deducir a partir de las fórmulas de primer orden de F las valuaciones a(valor) y e(argumentos))}. Donde $|S_A^n|$ son los n argumentos de un evento[2].

Es decir, proyecta cada uno de los mundos posibles en el conjunto de partes finitas de los pares (*atributo, valor_atributo*) y (*evento, valor_argumentos*).

Observemos que a partir de la función *prop* podemos deducir π de forma unívoca dado que $\pi(f)$ = { $w \in W$ | *podemos deducir f a partir de prop*(w)}. Esta simetría la expresaremos en adelante de forma simplificada como $w \in \pi(f) \Leftrightarrow f \in prop(w)$.

Definición 2.3 Dada una estructura S = (W, π), un elemento del process p ∈ P, y un mundo w ∈ W, entonces un mundo w' ∈ W es un **mundo p-successor** de w si (w, w') ∈ m(p). El conjunto de todos los mundos **p-successor** de w en una estructura S se escribe como successor(p)(w), es decir:

$$successor(p)(w) = \{w' \in W \mid (w, w') \in m(p)\} \quad (1)$$

Escribiremos S, w |= ϕ si w ∈ $\pi(\phi)$. Si ∀ w ∈ W: S, w |= ϕ lo notamos por S |= ϕ y |= ϕ (ϕ es válida) si para todas las estructuras S se cumple que S |= ϕ.

Definición 2.4 La función π se define inductivamente de la forma[3] [21] :

1) $\pi(\phi \vee \psi) = \pi(\phi) \cup \pi(\psi)$
2) $\pi(\neg\phi) = W \setminus \pi(\phi)$
3) $\pi([p]\phi) = \{w \in W \mid \forall w' \in W, (w, w') \in m(p) \Rightarrow w' \in \pi(\phi)\}$

Proposición 2.5 Para cada estructura S = (W, π), un mundo w ∈ W, fórmulas ϕ, ψ y p ∈ P deberá cumplirse que [21]:

1) $S, w \models \neg\phi \Leftrightarrow S, w \not\models \phi$
2) $S, w \models \phi \vee \psi \quad \Leftrightarrow S, w \models \phi \text{ or } S, w \models \psi$
3) $S, w \models true$
4) $S, w \not\models false$
5) $S, w \models \phi \wedge \psi \quad \Leftrightarrow S, w \models \phi \text{ and } S, w \models \psi$
6) $S, w \models \phi \rightarrow \psi \Leftrightarrow \text{if } S, w \models \phi \text{ then } S, w \models \psi$
7) $S, w \models \phi \leftrightarrow \psi \Leftrightarrow \text{if } S, w \models \phi \text{ iff } S, w \models \psi$
8) $S, w \models [p]\phi \Leftrightarrow \forall w' \in succ(p)(w')$: $S, w' \models \phi$

4. LÓGICA DINÁMICA PARA OASIS

En OASIS [14] se consideran los siguientes tipos de fórmulas en lógica dinámica correspondientes a determinadas secciones de la especificación de una clase.

[1] Ej., el conjunto soporte del sort de los números *Naturales*, es |*Naturales*|={0, 1, 2, ...}

[2] Es importante resaltar que aumentamos el concepto de estado con los eventos pendientes de ejecutarse.

[3] $[p]\phi$ expresa que tras p, ϕ es necesariamente cierta.

- **Evaluaciones.** Son fórmulas de la forma $\phi[p]\psi$ cuya semántica viene dada por la función m(p) que a partir del término p $\in$ P devuelve una relación entre posibles mundos. Es decir, dada una estructura S = (W, π), estando en el estado[4] w, antes de la ejecución de p se cumplirá que w $\in \pi(\phi)$ y, además, siendo w' $\in$ W un **mundo p-successor** de w, se cumplirá que w' $\in \pi(\psi)$.

- **Derivaciones.** Son fórmulas del tipo $\phi \rightarrow \psi$ que nos permiten definir atributos derivados (indicados en ψ) en términos de una condición de derivación (expresada en ϕ).

Difieren de las fórmulas de valuación en que las derivaciones relacionan un único estado, es decir, si dado $w \in W$, $w \in \pi(\phi) \Rightarrow w \in \pi(\psi)$.

- **Restricciones de Integridad.** Son fórmulas que deben cumplirse en cada mundo posible. Se distingue entre **estáticas** y **dinámicas** dependiendo de si su definición se expresa sobre el mundo actual o si relacionan varios mundos, respectivamente.

- **Precondiciones.** Son fórmulas del tipo $\neg\phi$[p]false. Estando en un mundo w, tal que si S, w $\models\phi$ y se produce p, entonces no existe un mundo w' alcanzable (dado que en ningún mundo se evalúa false).

La sintaxis de OASIS v.2 permite expresar la precondición anterior de la forma *p if* ϕ.

- **Triggers.** Son fórmulas de la forma $\phi[\neg p]$false. donde $\neg$p representa que no se lleva a cabo la acción indicada por p. Es decir, estando en un mundo w tal que S, w $\models\phi$, entonces no hay estados alcanzables para $\neg$p.

Con este tipo de fórmulas dinámicas forzamos la ocurrencia de las acciones, es decir, a que ocurra $(w, w') \in m(p)$ en el estado w si se cumple que $w \in \pi(\phi)$. La sintaxis de OASIS v.2 permite expresar en la sección de Triggers de la forma *p if* ϕ dada la fórmula $\phi[\neg p]$*false*.

Es importante resaltar que no se indica que no pueda ocurrir otra cosa distinta de p, de hecho, esto último debe ser posible, por ejemplo, para indicar el disparo más de un trigger en un estado. Es este caso, se tiene que $p = p_1 \parallel p_2 \parallel \ldots \parallel p_n$, ($\parallel$ es el operador de composición paralela del Álgebra de Procesos), donde los p_i , i=1, .., n ,son las son las acciones de los triggers $\phi_i[\neg p_i]$ *false,* tales que $S, w \models \phi_i$ i=1, .., n.

- **Process.** La estructura de Kripke definida puede extenderse, tal como hace Wieringa en [24], con el álgebra de procesos de manera que se interpreten los términos de proceso. Es decir, cada término del álgebra de procesos (que representa al proceso que es el objeto) determina una relación de accesibilidad sobre la estructura de Kripke.

De esta forma se puede redefinir la relación de alcanzabilidad:

$$m : T_{Process}(X) \rightarrow (\Sigma \rightarrow (W_1 \rightarrow W_2)) \qquad (2)$$

donde, $T_{Process}(X)$ es el conjunto de términos asociados al álgebra de procesos de la especificación, X es un conjunto de variables, Σ son asignaciones a dichas variables en los sorts respectivos, y W_1 , W_2 son conjuntos de mundos. De esta forma, siendo σ una posible asignación de variables, se llega a que:

$$S, w, \sigma \models [p]\phi \Leftrightarrow \forall p \in P \text{ se cumple } m(p)(\sigma)(w) \models \phi \quad (3)$$

En todas las fórmulas anteriores, ϕ y ψ son fórmulas bien formadas que siguen las siguientes reglas, tal como se define en [14]:

- **Términos.**

- Cada constante de un sort es un término de tal sort.
- Cada variable es un término del sort dado.
- Si $f : s_1 \times s_2 \times \ldots \times s_n \rightarrow s$ es una función, y $t_1, t_2, \ldots, t_n$ son términos, donde cada t_i pertenece al sort s_i, entonces $f(t_1, t_2, \ldots, t_n)$ es un término del sort s.
- Cada atributo es un término del sort dado.
- Sólo son términos los que siguen estas reglas.

- **Átomos.**

- Las constantes *true* y *false* son átomos.
- Si t_1 y t_2 son términos del mismo sort S, y R es uno de los siguientes operadores relacionales (que están definidos para el sort S): =, <>, <, <=, >, >=, entonces $t_1 R t_2$ es un átomo.

- **Fórmulas bien formadas.**

- Cada átomo es una fórmula bien formada.
- Si ϕ *y* ψ son fórmulas bien formadas, entonces *not* ϕ, ϕ *and* ψ, ϕ *or* ψ, son también fórmulas bien formadas con el significado lógico clásico.
- Sólo son fórmulas bien formadas las que se construyen con estas reglas.

[4] Utilizaremos el concepto de mundo equivalente al de estado

5. EJEMPLO OASIS

Incluimos un ejemplo típico de especificación OASIS:

class Account
identification
number:(number).
constant_attributes
number:**nat**; name:**string**.
variable_attributes
balance:**real**(0); times:**nat**(0); rank:**nat**(0).
derived_attributes
good_balance:**bool**.
private_events var N:**nat**.
open **new**; close **destroy**;
deposit(N); withdraw(N);
pay_commision; change_rank(N).
valuation
balance=M[deposit(N)]balance=M+N and times=times+1;
balance=M[withdraw(N)]balance=M-N and times=times+1;
balance=M[self:pay_commision]balance=M-0.5;
[self::pay_commision] times=0;
[change_rank(N)] rank=N.
derivation
good_balance={balance>=100}.
preconditions
withdraw(N) **if** balance>=N or
(balance<N and rank=2);
close **if** balance=0.
triggers
self::pay_commision **if** times=5 and
good_balance=false and rank=0.
processes
CUENTA = open.CUENTA0;
CUENTA0 = close +
deposit(N).CUENTA0 + withdraw(N).CUENTA0 +
pay_commision.CUENTA0 + change_rank(N).CUENTA0.
end_class

5.1. CONJUNTO DE ATRIBUTOS Y EVENTOS

Conjunto de Atributos A

A = {"number", "name", "balance", "times", "rank", "good_balance"}; S = {"nat", "string", "real", "bool"}

$sort_{number}$ = nat $\in S$; $sort_{name}$ = string $\in SORT$

$sort_{balance}$ = real $\in S$; $sort_{times}$ = nat $\in SORT$

$sort_{rank}$ = nat $\in S$; $sort_{good\_balance}$ = bool $\in SORT$

Veremos para esta especificación OASIS las fórmulas asociadas considerando las definiciones anteriores.

Conjunto de Eventos E

El conjunto E de eventos para la especificación del ejemplo es el que se detalla[5]:

E = {"open", "close", "deposit", "withdraw", "pay_commision", "change_rank"}

$sort_{open}$ = nat × string ; $sort_{close}$ = ε

$sort_{deposit}$ = nat $\in S$; $sort_{withdraw}$ = nat $\in SORT$

$sort_{pay\_commision}$ = ε ; $sort_{change\_rank}$ = nat $\in SORT$

5.2. CONJUNTO DE AXIOMAS C

A continuación se detallan los axiomas en lógica dinámica correspondientes a las secciones de Valuaciones, Derivaciones, Precondiciones y Triggers.

Valuaciones

Analicemos para cada una de las valuaciones las propiedades asociadas desglosándolas en la Tabla 1 que se adjunta:

1) balance=M[deposit(N)] balance=M+N and times=times+1;
2) balance=M[withdraw(N)] balance=M-N and times=times+1;
3) balance=M[self:pay_commision] balance=M-0.5;
4) true [self::pay_commision]times=0;
5) true [change_rank(N)]rank=N.

Derivaciones

El único atributo derivado es good_balance la fórmula de derivación viene dada por:

good_balance ← {balance>=100}.

Esta fórmula debe ser desglosada en dos para expresarse de la forma $\phi \rightarrow \psi$:

balance>=100 → good_balance=true.
balance < 100 → good_balance=false.

Precondiciones

La Tabla 2 muestra en qué mundos son posibles las acciones que llevan asociada la precondición:

1) withdraw(N) **if** balance>=N or (balance<N and rank=2);

2) close **if** balance=0.

[5] Asumimos por defecto que el sort relativo al mecanismo de identificación de la instancia está presente en los argumentos del evento. El símbolo ε representa la cadena vacía.

Val.	*Estado[6] de Partida w*	*Estado de Final w'*	*Acción p*
1)	$w \in W$: S, w |= (balance=M)	$w' \in W$: S, w' |= (balance=M+N and times=times+1)	deposit(N)
2)	$w \in W$: S, w |= (balance=M)	$w' \in W$: S, w' |= (balance=M-N and times=times+1)	withdraw(N)
3)	$w \in W$: S,w|=(balance=M)	$w' \in W$: S, w' |=(balance=M-0.5)	self:pay_commision
4)	$w \in W$: S, w |= *true*	$w' \in W$: S, w' |= (times=0)	self::pay_commision
5)	$w \in W$: S, w |= *true*	$w' \in W$: S, w' |= (rank=N)	change_rank(N)

Tabla 1: Representación de Valuaciones

Prec	*Estado posible w*	*Acción permitida*
1)	$w \in W$: S, w |= (balance>=N or(balance<N and rank=2))	withdraw(N)
2)	$w \in W$: S, w |= (balance=0)	close

Tabla 2: Representación de Precondiciones

Trig	*Estado w de condición de disparo del Trigger*	*Acción que debe disparar*
1)	$w \in W$: S, w |=(times=5 and good_balance=false and rank=0)	self::pay_commision

Tabla 3: Representación de Trigger

Triggers

La Tabla 3 muestra qué transición entre mundos es una obligación cuando se dispara el trigger asociado a la especificación del ejemplo:

1) self::pay_commision **if** times=5 and good_balance=false and rank=0.

La fórmula dinámica del trigger expresa que $\phi[\neg p]$*false*. Debemos observar que el estado $w \in \pi(\phi)$, es tal que nada más llegar a dicho estado, el trigger debe ser disparado pues estar en w y no haber disparado p lleva a *false* (tal como se define en [14] y se contempla en [9][10][19]). En el estado previo a w es donde el objeto detecta la obligación del disparo.

Process

Para el ejemplo OASIS, si *pre* es la precondición del objeto que representa el proceso y *post* es la precondición postcondición, entonces se cumple que *pre*[*CUENTA*]*post* (donde la definición del término de procesos *CUENTA* que representa al objeto es definida de forma recurrente tal como se indica en la sección de procesos del ejemplo).

6. EVOLUCIÓN DE OBJETOS

En los puntos anteriores se han planteado algunas de las definiciones necesarias para expresar la evolución de objetos que son instancias de clases. Consideraremos una versión simplificada de OASIS en la que se incluyen eventos privados, valuaciones, derivaciones, precondiciones y triggers. La evolución que sufre un objeto desde el instante en que es creado ha sido expresada según el Modelo de Ejecución de forma detallada en [10]. Podemos resumir el ciclo que lleva a cabo un objeto en los siguientes pasos:

[6] En nuetro planteamiento podemos usar el concepto de *estado* de forma equivalente al de *mundo*.

1) Seleccionar de la cola de eventos[7] un subconjunto consistente[8] que cumpla sus precondiciones.

2) Efectuar las evaluaciones asociadas a dichos eventos.

3) Comprobar las condiciones asociadas al disparo de triggers.

4) Volver al punto inicial para alcanzar el nuevo estado y poder entonces disparar los trigger calculados.

Es importante recordar que un objeto puede evolucionar no sólo por las acciones que el resto de la sociedad le demanda (perspectiva servidor), sino también por el hecho de pedir servicios al resto de objetos (perspectiva cliente).

Este hecho se asume en la Figura-1 en la que la acción p que lleva a cabo el objeto puede ser ofrecer o pedir un servicio:

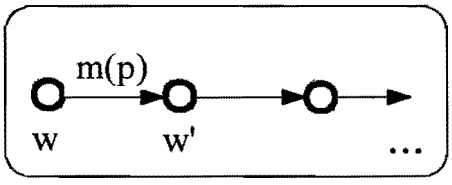


Figura-1: Transición entre estados de un objeto

Definición 4.1 El estado $w \in W$ en el que un objeto detecta la obligación de disparar el trigger expresado en la fórmula dinámica $\phi[\neg p]$false, se denomina **estado-detección** y viene determinado por:

- *Siendo $q \in P$ la acción que va a ejecutar, deberá cumplirse que $(w, w') \in m(q) \wedge w' \in \pi(\phi)$.*

Dicho de otro modo, el **estado-detección** de trigger es el estado previo al cual se cumple la condición de trigger.

7. TRADUCCIÓN A UN MODELO DE CONCURRENCIA. LA FUNCIÓN h

Dado que los lenguajes de descripción de sistemas tienden a representar abstracciones de sistemas más que sistemas concretos, el significado de una expresión en dichos lenguajes se da de forma general como *clases de equivalencia* en un nivel de abstracción concreto. Modelos de Concurrencia tales como Redes de Petri, Estructuras de Eventos, Árboles de Sincronización y los Sistemas de Transición-Estado modelan, esencialmente, un *proceso* como una evolución de estados, consecuencia de la ocurrencia de eventos.

Estamos interesados en representar la evolución de la sociedad de objetos de OASIS utilizando como modelo semántico las Redes de Petri Objetuales (OPNs)[8][6][5] [7] tal como hemos hecho en [19] o con Programación Lógica Concurrente (PLC) tal como se hace en [11].

Evidentemente, es necesario justificar de manera formal la traducción que de OASIS se haga a las OPNs o a PLC. Por simplificar el problema, en un primer estudio se abordará únicamente la perspectiva básica de un objeto, como entidad autónoma y aislada que de forma concurrente convive con el resto de la sociedad. Definiremos, dada una representación de nuestra sociedad de objetos basada en teorías de la lógica dinámica, una función de traducción a las redes de Petri objetuales OPN. Tratamos de verificar que la evolución que sufra el objeto visto como teoría, es equivalente a la que tenga como red de Petri o como cláusulas concurrentes.

Es decir, se desea establecer las bases para que el diagrama de la Figura 2 conmute. Tal como se muestra en la Figura 2, dada una teoría que represente a un objeto en un estado w, la ocurrencia de $p \in P$ implicará la transición al nuevo estado w' dado que $(w,w') \in m(p)$.

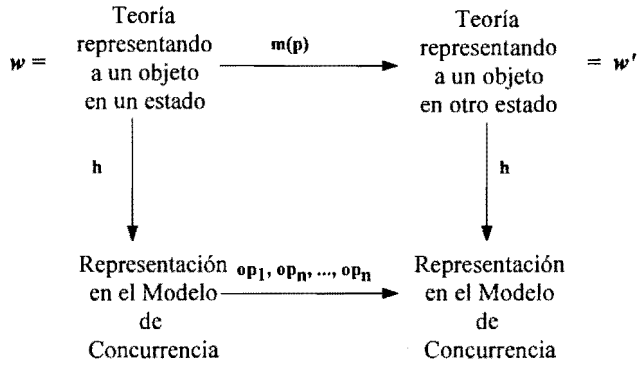


Figura-2: Diagrama de equivalencia

Por otro lado, aplicamos una **función de transformación h** desde la teoría a la representación que se elija para el modelo de concurrencia escogido. Cada operación $p \in P$ en el nivel superior equivaldrá a una o más operaciones op_1, op_2, ..., op_n realizadas en el nivel inferior.

8. REPRESENTACIÓN DE OBJETOS EN TEORÍAS

Tal como se indicó en la definición 2.2, el estado (o mundo) en el que se encuentra un objeto esta representado por la valuación de sus atributos y el conjunto de eventos pendientes de atender. Esto se definió a través de la función $prop(w) = (_{a \in A}\{a\} \times \{|sort(a)|\}) \cup (_{e \in E}\{e\} \times \{|sort^n(e)|\})$. Consideraremos un lenguaje L de la Teoría Clausal de primer orden para expresar el valor de los atributos de la forma *atributo(valor)* y para representar los eventos pendientes de ejecutarse. Este lenguaje se define como sigue:

[7] En el Modelo se contempla el que todo objeto tiene un buffer no acotado de eventos pendientes de atender.

[8] Básicamente, que no actúen sobre el mismo conjunto de atributos.

- **Términos**

- Un elemento del conjunto soporte de un sort S_a, $\forall a \in A$, es un término de dicho sort. (Ej. $t \in |sort(a)|$).
- Sólo son términos los anteriores.

- **Átomos**

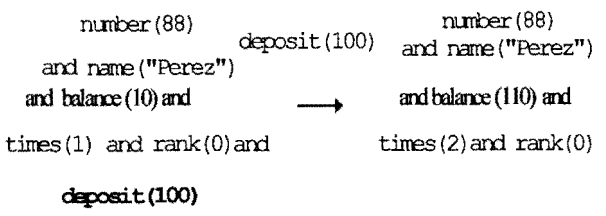
- Si $a \in A$ es un atributo y t es un término del soporte $\in |sort(a)|$, entonces $a(t)$ es un átomo.
- Si $e \in E$ es un evento de aridad n con argumentos t_1, ..., t_n que son términos del conjunto soporte S_a^i, i=1,..., n, respectivamente, entonces $e(t_1, ..., t_n)$ es un átomo.
- Sólo son átomos los obtenidos a través de las reglas anteriores.

- **Fórmulas bien Formadas (fbfs) del lenguaje L**

- Un átomo de L es una fbf.
- Si F, G son fbfs, entonces F *and* G es una fbf.\
- Sólo son fbfs de L las obtenidas a través de las anteriores reglas.

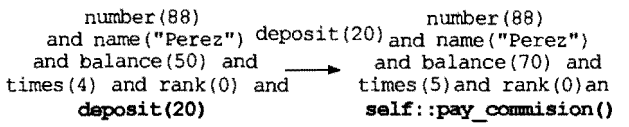
Ejemplo de estado

Un estado para un objeto instancia de la clase Account del ejemplo anterior y expresado como valuación de los atributos podría ser[9] el que se indica a la izquierda y tras el evento deposit(100) pasar a ser el de la derecha:

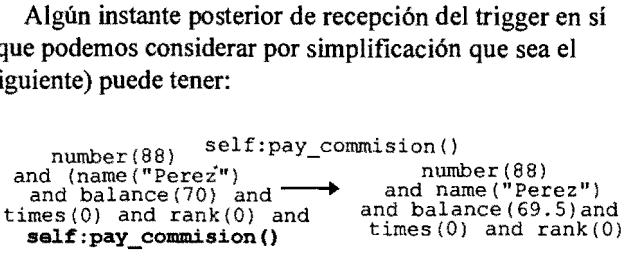


Esta consideración nos ha llevado a incluir como parte del estado la bolsa de eventos pendientes de ejecutar ya que permite representar tanto el hecho de disparar como el que el evento llegue a estar en la cola del servidor del evento. Un *estado-detección* puede ser el que se incluye a continuación en la parte izquierda. El estado al que llega el objeto tras el evento deposit(20) es el de la derecha. Se destaca el tener como evento pendiente de atender self::pay_commision (en algún estado posterior recibirá el trigger self:pay_commision como tal).

[9] Para una ejecución completa de una traza referirse a [12].



Algún instante posterior de recepción del trigger en sí (que podemos considerar por simplificación que sea el siguiente) puede tener:



En este ejemplo, la repercusión del disparo del trigger sólo influye en la teoría del propio objeto. En caso de que el servidor del trigger no sea la propia instancia, el hecho de disparar el trigger conlleva cambios en dos teorías: (1) en la relativa al objeto cliente del servicio, y (2), en la del objeto servidor del mismo.

9. TRADUCCIÓN A REDES DE PETRI

Las redes de Petri modelan los procesos en términos de cómo la ocurrencia de **eventos** *provoca cambios en los estados locales llamados* **condiciones** [25]. Esto se expresa mediante una dependencia causal entre los conjuntos de eventos y los conjuntos de condiciones y es ésta la que determina la conducta dinámica de las redes. En el DSIC-UPV estamos trabajando [19][20] con las mencionadas OPNs [8] por ser un tipo de redes de Petri de alto nivel que incorporan, entre otros, el concepto de *clase*, el mecanismo de *herencia*, el *polimorfismo*, el *enlace dinámico*, arcos inhibidores y de test. Para nuestro objetivo, contemplaremos por ahora el tipo más simplificado y general de redes de Petri tal como se introducen en [25].

9.1. MODELO GENERAL DE REDES DE PETRI

Introduciremos, según se desarrollan en [25], los conceptos necesarios para poder entender la formalización y el proceso de traducción a Redes de Petri.

Definición 9.1 Una **Red de Petri** es una cuatro-tupla (B, E, F, M_0) donde:

1. *B es un conjunto de* **condiciones**,
2. *E es un conjunto de* **eventos**,
3. *F is a multiset of* $(B \times E) \cup (E \times B)$, *llamada* **relación de dependencia causal**,
4. M_0 *es un multiconjunto no nulo de condiciones, llamado* **marcado inicial.**

que satisface las siguientes restricciones,

$\forall e \in E\ \exists b \in B: F_{b,e} > 0$ and $\forall e \in E\ \exists b \in B: F_{e,b} > 0$. (3)

Las redes de Petri se dibujan como grafos en los cuales los eventos (transiciones) se representan como rectángulos y las condiciones (lugares) como círculos con arcos dirigidos entre ellos y pesos asociados con números enteros positivos para representar la relación de flujo. El marcado inicial se representa mediante tokens con la multiplicidad apropiada en cada condición.

9.2. REDES VISTAS COMO ÁLGEBRAS

Es útil, tanto notacional como conceptualmente, ver una Red de Petri [25] como una **2-sorted** álgebra sobre multiconjuntos (entre otras cosas, facilita el cálculo de invariantes).

Las redes tienen una correspondencia 1-1 con una 2-sorted álgebra siendo los sorts μB un multiconjunto de condiciones y μE un multiconjunto de eventos, la constante $M_0 \in \mu B$, y las operaciones unarias siguientes:

$$\bullet(-): E \to \mu B \qquad y \qquad (-)\bullet: E \to \mu B$$

Donde $\bullet(-)$ establece la precondición (en cuanto a disponibilidad de los recursos indicados en la función) para que ocurra el evento que es su primer argumento. De forma análoga $(-)\bullet$ representa las postcondiciones o recursos generados por la ocurrencia del evento indicado como argumento.

que satisfacen $M_0 \neq \underline{0}$ *and* $((\bullet A = \underline{0}$ *or* $A\bullet = \underline{0}) \Rightarrow A = \underline{0})$,

es decir, la ocurrencia de cualquier evento necesita obligatoriamente consumir y generar recursos.

Evolución Dinámica de las Redes

Los estados de la red se representan mediante *marcados* que son multiconjuntos de condiciones. Típicamente, las condiciones pueden verse como recursos y su multiplicidad como la cantidad de dicho recurso. Cuando un evento se ejecuta consume determinados recursos y produce otros. Esta cantidad viene determinada por la relación F (véase Def. 9.1). De esta manera, si existen recursos suficientes entonces más de un evento puede ocurrir concurrentemente, y, hasta es posible que un evento pueda ocurrir con determinada multiplicidad. Por simplificar el planteamiento se considerará una subclase de las anteriores redes llamadas **contact-free** o **safe**, en las que esta multiplicidad no puede darse.

Sea $N = (B, E, F, M_0)$ una red de Petri. Un marcado M es un multiconjunto de condiciones, es decir, $M \in \mu B$.

Sean M, M' marcados. Sea A un multiconjunto finito de eventos. Se define[10]:

$$M \xrightarrow{A} M' \Leftrightarrow \bullet A <= M \text{ and } M' = (M - \bullet A) + A\bullet \quad (4)$$

Esto nos da la *relación de transición* entre los marcados. Cuando queremos enfatizar la red N en la que se da la transición $M \xrightarrow{A} M'$ escribimos:

$$N: M \xrightarrow{A} M' \qquad (5)$$

Un *marcado alcanzable* de N es aquel marcado M para el cual se cumple que existen sucesivos A_i, i=0, .., n−1:

$$M_0 \xrightarrow{A_0} M_1 \xrightarrow{A_1} \circ\circ\circ \xrightarrow{A_{n-1}} M_n = M \quad (6)$$

Definición 9.2 *Una Red de Petri* $N = (B, E, F, M_0)$ *es* **contact-free** $\Leftrightarrow F <= \underline{1}$ y $M <= \underline{1}$ para todos los marcados alcanzables M.

Para este tipo de redes, una condición es cierta o no para que un evento ocurra, pero sin multiplicidad (redes *safe*).

Proposición 9.3 El comportamiento de una red de Petri $N = (B, E, F, M_0)$ **contact-free** se define como:

1. *A es un conjunto de* **eventos**. Entonces, $\bullet A$ y $A\bullet$ son conjuntos también.
2. Cualquier marcado alcanzable es un conjunto.
3. Siendo M un marcado alcanzable y siendo M' otro marcado de N, entonces :

$$M \xrightarrow{A} M' \Leftrightarrow (\forall e \in A.\ \bullet e \subseteq M) \text{ and}$$

$$(\forall e, e' \in A.\ \bullet e \cap e\bullet = \varnothing) \text{ and } M' = (M \setminus \bullet A) \cup A\bullet \quad (7)$$

Para un red **contact-free**, un evento e se dice que está **habilitado** en un marcado M si $\bullet e \subseteq M$. Si dos eventos e y e' están habilitados en un marcado M y comparten una precondición común, es decir, $\bullet e \cap e\bullet \neq \varnothing$, entonces los eventos e y e' se dice que están en **conflicto** en M. Por otro lado, si tenemos $M \xrightarrow{A} M'$ entonces los eventos en A se dice que ocurren de forma **concurrente**.

9.3. DE OBJETO OASIS A REDES DE PETRI

En [19] hemos visto unas guías para representar los aspectos básicos de comunicación y estructurales de OASIS en las OPNs. Como mencionamos en puntos anteriores, restringiremos el análisis únicamente a la perspectiva individual del objeto. Para exigir que el diagrama de la Figura-2 conmute, vamos a utilizar las redes introducidas en el punto anterior y más tarde podremos extender el estudio a las OPNs.

Previo al proceso de representación, necesitamos de las siguientes definiciones.

[10] Se considera que el lector interpreta las operaciones sobre multiconjuntos. Para mayor detalle, dirigirse a [17].

Definición 9.5 Siendo N = (B, E, F, M_0) una red de Petri representando un objeto instancia de una clase ∈ especificación OASIS y siendo $M_0 \xrightarrow{A_0} M_1 \xrightarrow{A_1} \circ\circ\circ \xrightarrow{A_{n-1}} M_n$ una secuencia de marcados alcanzables, se define un **marcado-estable M_i** el que representa un **nuevo estado** del objeto que se está representado en dicha red, de manera que ∀ j < i, M_j corresponde a un estado anterior a dicho estado.

Con esta última definición se establece una relación entre los posibles estados del objeto en la Teoría con los marcados en la Red de Petri. En la Figura-9.4 vemos que los estados posibles de un objeto tienen correspondencia un subconjunto de los marcados de la red de Petri (los que llamamos *marcados-estables*):

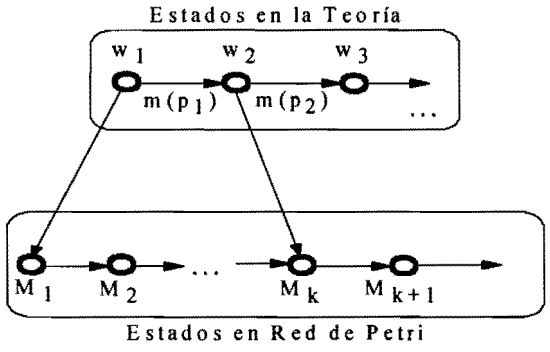


Figura-9.4: Correspondencia entre estados.

Definición 9.6 Siendo N = (B, E, F, M_0) una red de Petri para una especificación OASIS y M_i un **marcado-estable** de dicha red, se define la función **$prop_N$**:

$M_i \rightarrow ((A \times S_A) \cup (E \times S^r_A)$ como $prop_N(n, M_i) = \{(_{a\in A}\{a\} \times |sort(a)|\ ) \cup (_{e\in E}\{e\} \times |sort^r(e)|\ )\}$. Donde n ∈ N, S^r_A son los r argumentos de un evento con el sort asociado ∈ A.

Definición 9.7 Una **función de transformación**[11] **h** es la que asocia a cada estado del objeto su correspondiente marcado-estable.

Proposición 9.8 Decimos que una red de Petri N = (B, E, F, M_0) **satisface** el Modelo de Ejecución [10] de una Especificación OASIS si dada una función de transformación **h** : W → μB y ∀ p ∈ P, se cumple que:

(i)Mi = h(wk) ⇒ propN(Mi) = prop(w) ∀wk∈W, Mi∈M.

(ii)Mi = h(wk) ⇒ Mj = h(wk+1) ∀ wk , wk+1 ∈W, p ∈ P,

siendo w_{k+1} un mundo **p***-succesor de w_k , M_j ∈ M el marcado-estable sucesor de M_i*

Es decir, hablar del estado desde la teoría que representa al objeto es equivalente a hablar del estado desde la representación que en el modelo de concurrencia de las Redes de Petri. Una vez se haya obtenido una red candidata a ser solución de la especificación OASIS debemos probar que los marcados que alcanza la red de Petri cumple las restricciones anteriores ∀ *p* ∈ *P*. Los formalismos matemáticos definidos sobre ellas permiten el poder calcular analíticamente si se cumplen las propiedades especificadas.

10. CONCLUSIONES

La obtención de prototipos de forma automática que ayuden a la validación temprana de especificaciones es un aspecto importante dentro de las nuevas tendencias en la ingeniería del software. Otro aspecto clave es la utilización de lenguajes formales que permitan la validación de los sistemas especificados de manera formal. Es obvio que las herramientas formales no son agradables al usuario que especifica y en general requiere una formación excesiva para poder abarcarlas. Por otro lado, es deseable obtener un soporte formal que ayude a verificar que el sistema que está especificando es el que realmente se necesita. De esta forma, el usuario no tiene porqué saber que el prototipo que está ejecutando *satisface* de manera formal su especificación; eso es tarea de quien investiga, de quien fija las pautas de construcción de los prototipos. Lo que hemos descrito en este trabajo puede ser extrapolado a la programación lógica concurrente, a otros tipos de redes de petri, y en general, al resto de modelos de concurrencia siempre que sea posible establecer la función de transformación *h* correspondiente.

Seguimos trabajando buscando:

- Incluir aspectos de comunicación más complejos tales como eventos compatidos.
- Considerar estructuras complejas de clases incluyendo la herencia y la agregación de OASIS.

[11] Ver Figura 2.

11. REFERENCIAS

[1] Dubois E., Du Bois Ph., Petit M., *ALBERT: An Agent-Oriented Language for Building and Eliciting Requirements for Real-Time Systems* in Proc. of the 27th HICSS, J.F. Nunamacker and R.H. Sprague (eds), IEEE Computer Society Press, Vol. 4, pp. 713-722, Jaunary 1994.

[2] Dubois E., Du Bois P., Dubru F., *Animating Formal Requirements Specifications of Cooperative Informa-tion Systems* in Proc. of the Second International Conference on Coopera-tive Information Systems, Toronto (Canada), May 1994.

[3] Hartel P., Denker G., Kowsari M., Krone M., Erich H.D., *Information Systems Modelling with TROLL Formal Methods at work*, Information Systems Vol.22, No.2/3, pp.79-99, 1997.

[4] Hartmann T., Saake G., Jungclaus R., Hartel P., Kusch J., *Revised Version of Modelling Language TROLL*, Infor-matik-Bericht 94-03, TU Braunsch-weig, 1994.

[5] Keen C., *Lakos C. Information Modelling using LOOPN++, an Object Petri Net Scheme*, Proceedings of 4th International Working Conference on Dynamic Modelling and Information Systems, 1994.

[6] Lakos C., Keen C., *LOOPN++: A new Language for Object-Oriented Petri*, Proceedings of Modelling and Simulation, Barcelona, Society for Computer Simulation, 1994.

[7] Lakos C., Keen C., *An Open Software Engineering Environment Based on Object Petri Nets*, Department of Computer Science, University of Tasmania, 1995.

[8] Lakos C., *From Coloured Petri Nets to Object Petri Nets*, Proceedings of 16th International Conference on the Application and Theory of Petri Nets, LNCS 935, Torino, Italy, Springer-Verlag, 1995.

[9] Letelier P., Sánchez P., Ramos I., *Animación de Modelos Conceptuales en un Entorno Concurrente*, Informe Técnico, DSIC, Universidad Politécnica de Valencia, 1997.

[10] Letelier P., Sánchez P., Ramos I, *Animación de Modelos Conceptuales para ayudar en la Validación de Requisitios*, ASOO'97, Buenos Aires, 1997 August.

[11] Letelier P., Sánchez P., Ramos I., *Animation of Systems Specifications Using Concurrent Logic Programming*, Symposium in Logical Approaches to Agent and Modelling and Design, ESSLLI'97, Aix-en-Provence, France, 1997, August. (accepted).

[12] Letelier P., Sánchez P., Ramos I., *Guías para la Generación Automática de un Programa Lógico Concurrente que sirve de prototipo para una especificaión OASIS*, Informe Técnico, DSIC, Universidad Politécnica de Valencia, 1997.

[13] Pastor O. *Diseño y Desarrollo de un Entorno de Producción Automática de Software basado en el modelo orien-tado a Objetos*, Tesis doctoral dirigida por Isidro Ramos, 1992, DSIC Universidad Politécnica de Valencia.

[14] Pastor O., Ramos I., *OASIS versión 2 (2.2) : A Class-Definition Language to Model Information Systems Using an Object-Oriented Approach*, SPUPV-95.788, Universidad Politécnica de Valencia, 1995.

[15] Pastor O., Barberá J.M., Merseguer J., Romero J., Insfrán E., *The CASE OO-METHOD graphic environment description*. Tech. Report, ITI-DT-96.

[16] Pastor O., Insfrán E., Pelechano V., Romero J., Merseguer J., *OO-METHOD: An OO Software Production Environment Combining Conventional and Formal Methods*. CAiSE97. Barcelona Spain, June 1997.

[17] Pelechano V., Pastor O., *Case OO-Method : Un entorno de producción automática de software orientado a objetos*, Proc.of CIL-95, Barcelona, 1995.

[18] Ramos I., Pastor O., Cuevas J., Devesa J., *Objects as Observable Processes*, Proceedings of the 4th Workshop on the Deductive Approach to Information Systems Design, Catalonia, 1993.

[19] Sánchez P., Letelier P., Ramos I., *Constructs for Prototyping Information Systems using Object Petri Nets*, IEEE System Man and Cybernetics, Orlando, October 1997. (accepted)

[20] Sánchez P., Ramos I., *Object Petri Nets: Modelo de Implementación para OASIS*, Technical Report DSIC-II/17/96, Universidad Politécnica de Valencia.

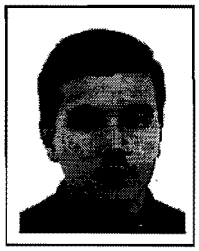
[21] Spruit P., Wieringa R., Meyer J-J., *Axiomatization, declarative semantics and operational semantics of passive and active updates in logic databases*, Department of Mathematics and Computer Science, Vrije Universiteit, Amsterdam.

[22] Wieringa R.J., *A Conceptual Model Specification Language (CMSL Version 2)*, Technical Report IR-248, Vrije Universiteit, Amsterdam, 1991.

[23] Wieringa R.J., *A Formalization of Objects using Equational Dynamic Logic*, 2nd Conference on Deductive and OO Databases, pags 431-452, Springer 1991, Lecture Notes in Computer Science 566.

[24] Wieringa R.J., Meyer J.-J.Ch, *Actors, Actions, and Initiative in Normative System Specification*, Annals of Mathematics and Artificial Intelligence, 7:289-346, 1993.

[25] Winskel G., *Categories of Models for Concurrency*, LNCS 197, pp.246-267, 1984.



Pedro Sánchez Palma. *Nacionalidad española, es licenciado en informática, por la Universidad de Málaga, España. Sus áreas de interés son: modelización conceptual de sistemas de información, orientación a objetos, ingeniería de la programación, validación de requirimientos y redes de Petri de alto nivel (Object Petri Nets). Actualmente trabaja como profesor en el departamento de sistemas informáticos y computación en la Universidad de Politécnica de Valencia, Valencia , España y esta desarrollando su tesis doctoral en las áreas antes mencionadas.*

Patricio Letelier Torres. *Nacionalidad chilena, es ingeniero civil en informática, egresado de la Universidad Técnica Federico Santa María, Valparaiso, Chile. Sus áreas de interés son: ingeniería de requisitos, modellización orientada a objetos, animación de modelos conceptuales y programación lógica concurrente. Actualmente trabaja como profesor asociado en el departamento de sistemas infomáticos y computación de la Universisdad Politécnica de Valencia, España y esta desarrollando su tesis para la obtención del grado de Doctor en Informática.*

Isidro Ramos Salavert.*Nacionalidad española, obtuvo el grado de Doctor en Ciencias. Su área de interés: modelización conceptual de sistemas de información. Actualmente es catedrático de Universisdad, director de grupo de Programación Lógica e Ingeniería de Software de la UPV (DSIC).*