Formalización de OASIS en Lógica Dinámica incluyendo especificaciones de proceso

P.Letelier, P.Sánchez, I.Ramos, O.Pastor
Departament de Sistemes Informàtics i Computació
Universitat Politècnica de València

Camì de Vera, s/n, 46071 València
☎ 3877350 ; Fax :3877357
email {letelier | ppalma | iramos | opastor}@dsic.upv.es

Resumen

OASIS en su versión 2.2 de OASIS utiliza como marco formal una variante de Lógica Dinámica. Sin embargo, la sección de especificación de procesos en la plantilla de una clase es formalmente tratada mediante un Algebra de Procesos para objetos. Este trabajo presenta una nueva visión para especificaciones de procesos en OASIS. Desde una perspectiva deóntica una especificación de clase en OASIS es un conjunto de axiomas de prohibición y obligación de ocurrencia de acciones, además de un conjunto de axiomas que establecen los mundos previos e inmediatamente posteriores a la ocurrencia de una acción. Consecuentemente, se distinguen dos tipos de especificaciones de procesos: aquellos que establecen prohibiciones (*patterns*) y aquellos que establecen obligaciones (*operations*). Se muestra cómo una especificación de procesos básica, incluyendo operadores de secuencia y alternativa, puede ser vista como un conjunto de axiomas en Lógica Dinámica. Esto permite formalizar toda la plantilla de clase OASIS en Lógica Dinámica. Utilizando un sublenguaje de CCS se presentan las correspondencias entre especificaciones de proceso en dicho lenguaje y axiomas en Lógica Dinámica. Se incluyen algunos ejemplos aplicados a OASIS.

1 Introducción

La Lógica Dinámica ofrece una formalización directa e intuitiva para especificaciones OASIS. Así, las fórmulas para precondiciones, disparos, evaluaciones, derivaciones y restricciones de integridad constituyen un azúcar sintáctico de axiomas en lógica dinámica. Sin embargo, hasta la versión 2.2 de OASIS, la sección para especificación de procesos es tratada desde otra perspectiva formal: la teoría de procesos, utilizando un Álgebra de Procesos para Objetos (APO).

Desde la perspectiva de este trabajo, hay tres aspectos de los procesos modelados en OASIS 2.2 en los cuales se centrará el estudio:

- En las especificaciones de proceso no se incluyen los servicios requeridos por el objeto, es decir, no se considera la perspectiva cliente del objeto.

- Cada clase tiene una única especificación de proceso que incluye a todos las acciones que pueden ser servidas por el objeto.

- Las especificaciones de procesos sólo tienen un carácter restrictivo (de prohibición) sobre las vidas posibles del objeto.

En Figura 1.1 se representa a un objeto como un rectángulo que en su interior contiene la especificación de proceso del objeto expresada en un diagrama de transición de estados. Los rectángulos del costado derecho son conectores asociados a servicios provistos (rectángulos negros) y a servicios solicitados (rectángulos grises). Las transiciones del diagrama están asociadas a la ocurrencia de una acción de solicitar o de proveer un servicio.

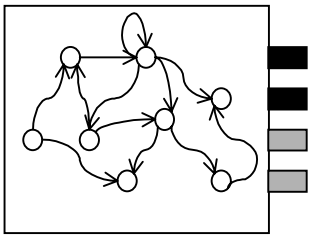

Figura 1.1: Representación de un objeto

La especificación de proceso de la clase debe ser consistente con el resto de la plantilla de la clase. Conseguir especificar el comportamiento de los objetos de una clase usando sólo una especificación de proceso (sólo un diagrama de transición de estados) es en la mayoría de los casos muy difícil. Las precondiciones y disparos en general hacen que la especificación de procesos se haga más extensa y compleja. Las precondiciones hacen necesario incluir transiciones por no-satisfacción precondiciones, en OASIS 2.2 éstas existen de forma implícita y todas ellas conducen al mismo estado. Pero la principal dificultad radica en lograr incorporar la perspectiva cliente en una especificación de proceso, en particular al considerar la actividad cliente originada por disparos. Veamos el siguiente ejemplo en OASIS 2.2: una especificación preliminar para la clase socio de una biblioteca.

```
class socio
...
variable attributes
  num_libros: nat (0);
  notificado:bool (false);
private events
  alta new;
  baja destroy;
  prestar;
  devolver;
  notificar.
valuations
  [prestar] num_libros=num_libros+1;
  [devolver] num_libros=num_libros-1;
  [notificar] notificado=true.
preconditions
  prestar if num_libros<5.
triggers
  notificar if num_libros=1.
```

```
processes
  C  =     alta.C0;
  C0 =     baja + prestar.C1;
  C1 =     prestar.C1 + {num_libros>1}devolver.C1 + {num_libros=1}devolver.C0;
end_class
```

Los estados de una especificación de proceso están asociados con las propiedades de los objetos pero no necesariamente tiene una relación 1-1 con ellas. En el ejemplo, los estados de la especificación de proceso son C, C0, C1 y uno adicional implícito para representar la destrucción después de la acción baja. Si las propiedades de interés son los atributos num_libros y notificado, en C0 el valor de num_libros es 0 y notificado puede ser true o false. Asimismo, en C1 el valor de num_libros es un natural mayor o igual a 1 y notificado puede ser true o false.

De acuerdo con la precondición prestar if num_libros<5, todo estado que tenga alguna transición con la acción prestar y en el cual pueda darse num_libros>5, tiene una transición implícita con la acción reject_prestar, que representa el rechazo de prestar y cuyo estado alcanzado es el mismo. Esto sólo ocurre en el estado C1.

En la especificación de proceso no se ha usado la acción notificar que corresponde a un disparo. Los disparos junto a las definiciones de cliente (al etiquetar eventos en la especificación de la clase servidora) son los mecanismos que en OASIS 2.2 determinan la perspectiva cliente de un objeto. En el ejemplo anterior, incorporar la acción notificar a la especificación de proceso significa añadir una transición con dicha acción desde cada estado en cual pueda satisfacerse la condición de disparo (num_libros=1). En este pequeño ejemplo es sencillo pues sólo en C1 el num_libros puede tener el valor 1. Pero la transición añadida debe tener una guarda pues en el mismo estado el atributo num_libros puede tener otros valores diferentes a 1. Si la relación entre los estados de la especificación de proceso y los estados del objeto (vistos como el conjunto de valores de las propiedades del objeto) fuera 1-1, no sería necesario el uso de dichas guardas.

Si se quiere incorporar de forma total y homogénea la perspectiva cliente, entonces la acción de solicitar un servicio también podría tener evaluaciones asociadas. Esto significa que quizá al incorporar la perspectiva cliente en la especificación de proceso además de añadir transiciones por solicitar, probablemente deberían añadirse nuevos estados. Finalmente, si se considera que un objeto puede solicitar más servicio en un mismo instante (porque se satisface la condición de más de un disparo o porque se trata de un objeto compuesto), entonces la incorporación de la perspectiva cliente en la especificación de proceso de OASIS 2.2 se hace muy difícil.

Por otra parte, como se mencionó anteriormente, una especificación de proceso en OASIS 2.2 sólo es de carácter restrictivo respecto de las vidas posibles del objeto. Cuando se modela un proceso que expresa el conjunto de acciones que el objeto debe ejecutar por iniciativa propia el carácter necesario es de obligación en lugar de restricción. La capacidad expresiva para modelar estas secuencias de acciones, encontradas en forma de algoritmos y métodos en programación tradicional, en OASIS 2.2 sólo es posible con la definición de transacciones. El concepto de transacción es, sin embargo, más específico y que conlleva consideraciones adicionales tales como atomicidad y no observabilidad intermedia.

Los objetivos de este trabajo son dos: replantear el concepto de proceso (especificación de proceso) dentro de la plantilla de clase OASIS y al mismo tiempo proporcionar una correspondencia para dichas especificaciones de proceso dentro del marco de la Lógica Dinámica.

El concepto de proceso será enriquecido en los siguientes aspectos:

- Una especificación de procesos establece relaciones de alcanzabilidad entre estados, estas podrán ser por ocurrencia de acciones que el objeto recibe (perspectiva servidor) o que el objeto solicita (perspectiva cliente). Es decir, se considera la perspectiva cliente en la especificación de procesos.

- Los procesos especificados podrán tener una interpretación de prohibición o de obligación. Como veremos, esto último permite modelar tanto transacciones como métodos en sentido tradicional de POO. Así, el comportamiento de un objeto está condicionado por la prohibición u obligación de ocurrencia de ciertas acciones (sean ellas recibidas o solicitadas).

- Una clase podrá tener varias especificaciones de proceso (de prohibición o de obligación). La composición de las especificaciones de proceso se obtiene mediante su traducción a axiomas en Lógica Dinámica. El conjunto de axiomas resultantes por especificaciones de proceso junto a los axiomas obtenidos del resto de la plantilla determina el comportamiento de los objetos de la clase.

2 Plantilla o Tipo

Una clase en OASIS se compone de un nombre de clase, una función de identificación para instancias de la clase (objetos) y un tipo o plantilla que comparten todas las instancias.

El mecanismo para nombrar objetos se basa en una función de identificación. Ésta proporciona un conjunto de identificadores pertenecientes a un determinado *sort* (género) o a un *sort* definido por el usuario. Estos se importan en la definición de la clase. Los más usados están predefinidos como: *int*, *nat*, *real*, *bool*, *char*, *string* y *date*. Representan números, valores lógicos, caracteres, cadenas y fechas en un formato particular, respectivamente. Se pueden añadir nuevos dominios (*domains*) en una especificación definiendo el correspondiente tipo abstracto de datos.

Un tipo es la plantilla que recoge las propiedades (estructura y comportamiento) compartidos por todos los objetos potenciales de la clase considerada. Sintácticamente, puede ser formalizado como una signatura y un conjunto de axiomas en una variante de Lógica Dinámica. La signatura incluye los *sorts*, funciones, atributos y eventos utilizados. Los axiomas en Lógica Dinámica corresponden a las definiciones hechas en las diferentes secciones de la plantilla de una clase.

Existen dos grupos de axiomas: fórmulas y procesos. Entre las fórmulas se incluyen: restricciones de integridad *(integrity constraints*), derivaciones (*derivations*), precondiciones (*preconditions*), evaluaciones (*valuations*), disparos (*triggers*). Los procesos incluyen: operaciones (*operations*) y patrones de conducta (*patterns*).

De acuerdo con los conceptos anteriores, una **plantilla** o **tipo** de una clase se define como una tupla <*Att*, *E*, Φ, *Processes*>, donde:

- *Att* es el alfabeto de atributos. $\forall att \in Att$, se tiene una función
  att : *sort de nombrado* $\rightarrow$ *sort de evaluación*

- *E* es el alfabeto genérico de eventos. $\forall e \in E$ podemos obtener $\underline{e} = \theta e$ siendo θ una substitución básica de los posibles argumentos del evento.

- Φ es un conjunto de fórmulas en diferentes lógicas según sea la sección del lenguaje donde se utilizan.

- *Processes* es el conjunto de procesos. Cada proceso se especifica usando un cálculo de procesos basado en un subconjunto del lenguaje propuesto en CCS[Mil89].

En la vida de un objeto todo evento es solicitado por un cliente a un determinado servidor, pudiendo ser el objeto en cuestión cliente, servidor o ambos para cada acción. Llamaremos **acción** a la tupla <cliente, servidor, evento>. Como consecuencia, asumiremos definido un conjunto *A* de acciones obtenido a partir de *E*, el cliente y el servidor del evento. Al igual que para los eventos, $\forall a \in A$ podemos obtener $\underline{a} = \theta a$ siendo θ una substitución básica de los posibles cliente, servidor y evento.

Las fórmulas Φ y el cálculo de procesos *Processes* serán descritos en los apartados siguientes. La Lógica de Predicados de Primer Orden es usada en varias secciones del lenguaje, por esto, comenzaremos definiendo cúales serán Fórmulas Bien Formadas (F.b.f.) en Lógica de Predicados de Primer Orden.

Términos

- Cada constante de un *sort* es un término de dicho *sort*.
- Cada variable es un término de un *sort* dado.
- Si $f : s_1 \times s_2 \times ... \times s_n \rightarrow s$ es una función, y t_1, t_2, ..., t_n son términos, donde cada t_i pertenece al *sort* s_i, entonces $f(t_1,t_2,...,t_n)$ es un término del *sort* *s*.
- Cada atributo es un término de su *sort*.
- Sólo son términos aquellos construidos siguiendo las reglas anteriores.

Átomos

- Las constantes **true** y **false** son átomos.
- Si t_1 y t_2 son términos del mismo *sort*, y *R* es uno de los siguientes operadores relacionales: =,<>,<,<=,>,>= (asumidos definidos para el *sort* considerado) entonces $t_1 R\ t_2$ es un átomo.
- Sólo son átomos aquellos construidos siguiendo las reglas anteriores.

Fórmulas bien formadas en Lógica de Predicados de Primer Orden (F.b.f.)

- Cada átomo es una F.b.f.
- Si ϕ y ϕ' son F.b.f., entonces **not** ϕ , ϕ **and** ϕ', ϕ **or** ϕ', son también F.b.f. con el significado lógico usual.
- Sólo son F.b.f. aquellas construidas siguiendo las reglas anteriores.

Globalmente y considerando su expresividad, OASIS está basado en Lógica Deóntica[Åqv84]. La Lógica Deóntica es la lógica de obligaciones. prohibiciones y permisos. Sea $a \in A$, O(a) representa la obligación de ocurrencia de a, F(a) representa la prohibición de ocurrencia de a y P(a) representa el permiso o habilitación de ocurrencia de a. Por definición, $P(a) \Leftrightarrow \neg F(a)$ es decir, "a está permitida si y sólo si a no esta prohibida".

En OASIS la posibilidad de modelar comportamiento reactivo está ligada a su capacidad para expresar obligaciones. Más adelante veremos como dicho comportamiento reactivo puede ser expresado sobre la base del estado (*state-driven*) y sobre la base de secuencias de acciones (*event-driven*).

En [Mey88] la Lógica Deóntica es descrita en términos de variante de Lógica Dinámica [Har84]. La definición de los operadores deónticos en términos de Lógica Dinámica es la siguiente:

$F(a) \Leftrightarrow [\ a\ ]\ false$ "a está prohibida"
$O(a) \Leftrightarrow [\ \overline{a}\ ]\ false$ "a es obligatoria"

Donde $\overline{a}$ indica que la acción a no es ejecutada. En estas fórmulas *false* es un átomo especial para representar un estado de violación del sistema. El significado intuitivo de éstas fórmulas es: una acción es prohibida si su ejecución conduce a un estado de violación; una acción es obligatoria si su no-ejecución conduce a un estado de violación.

La Lógica Dinámica es un formalismo natural para estudiar de forma simple y directa ciertas aserciones sobre programas. En Lógica Dinámica, la fórmula $[a]\phi$ se interpreta intuitivamente como "después de la ejecución de la acción a, ϕ debe satisfacerse", siendo a una acción y ϕ una F.b.f. en Lógica de Predicados de Primer Orden.

Sea ψ una F.b.f. en Lógica de Predicados de Primer Orden que caracteriza el estado del objeto inmediatamente antes de la ocurrencia de la acción a, utilizaremos el siguiente tipo de fórmulas en Lógica Dinámica:

$$\psi \rightarrow [a]\ \phi$$

El significado intuitivo: "si el objeto está en un estado en el cual ψ se satisface entonces inmediatamente después de la ejecución de la acción a, ϕ debe satisfacerse".

A continuación se presentan cada una de las secciones de una plantilla de clase en OASIS describiendo el sublenguaje utilizado en cada caso. Posteriormente se describe cómo cada una de dichas secciones se corresponde con axiomas en la variante de Lógica Dinámica que se ha mostrado.

3 Especificación de Fórmulas

Para describir las fórmulas usadas en cada sección de una plantilla de clase, consideraremos ϕ,ψ como F.b.f. en Lógica de Predicados de Primer Orden y $a \in A$.

a) Restricciones de Integridad (Integrity Constraints)

Son fórmulas bien formadas en Lógica Temporal usadas para representar restricciones de integridad. Obedecen a las siguientes reglas:

- Toda ϕ es una F.b.f. en Lógica Temporal interpretada como **always** ϕ.
- **always** ϕ, **sometimes** ϕ, **next** ϕ, ϕ **since** ϕ' y ϕ **until** ϕ' son también F.b.f. en Lógica Temporal.
- Sea T un periodo de tiempo expresado usando las unidades de tiempo usuales: **minutes**, **hours**, **days**, **weeks**, **months**, **years**, entonces **sometimes**$_{<=T}$ ϕ **since** ϕ' es una F.b.f. en Lógica Temporal.
- Sólo son F.b.f. en Lógica Temporal válidas para esta sección las construidas siguiendo las reglas anteriores.

Si ϕ es una F.b.f. en Lógica Temporal para restricciones de integridad, entonces desde la Lógica Dinámica, dicha fórmula es vista como un conjunto de axiomas del tipo:

$$[a]\ \phi\ ,\ \forall a \in A.$$

b) Derivaciones (*Derivations*)

Las fórmulas usadas en derivaciones (información derivada) son del tipo $\phi \rightarrow \phi'$. El significado es el usual "si ϕ se satisface entonces ϕ' se satisface". ϕ' es un átomo que expresa mediante una igualdad cómo se obtiene el valor del atributo derivado.

En Lógica Dinámica, las fórmulas para derivaciones pueden ser representadas por

$$[a]\ (\phi \rightarrow \phi'),\ \forall a \in A.$$

c) Precondiciones *(Preconditions)*

Son fórmulas en la variante de Lógica Dinámica antes presentada. Representan prohibiciones para la ocurrencia de acciones. Así, una precondición es definida por una fórmula como la siguiente:

$\psi \rightarrow [a]\ false$ "si ψ se satisface entonces *a* está prohibida"

Como se mencionó antes, los permisos (o habilitación de ocurrencia de acciones) constituyen una sub-especificación, es decir, si una acción no está prohibida, entonces está permitida.

Por comodidad en la especificación, ψ será una fórmula del tipo $\neg\phi$, pues resulta más natural expresar "si $\neg\phi$ se satisface entonces *a* es prohibido".

Así, la fórmula en Lógica Dinámica para precondiciones se escribe finalmente como:

$\neg\phi \rightarrow [a]\ false$ "si $\neg\phi$ se satisface entonces *a* está prohibida"

c) Evaluaciones *(Valuations)*

Estas fórmulas definen los cambios de estado ante la ocurrencia de acciones. Una evaluación es una fórmula como la siguiente:

$\psi \rightarrow [a]\ \phi$

En este caso, ϕestá construida sólo usando átomos con el operador relacional = y conectivas **and**. Con esta sintaxis es posible caracterizar el estado de un objeto después de la ocurrencia de una determinada acción.

d) Disparos *(Triggers)*

Los disparos permiten especificar comportamiento reactivo basado en el estado del objeto (*state-driven*). Un disparo es una obligación y se define con una fórmula como la siguiente:

$\psi \rightarrow [\ \overline{a}\ ]\ false$ "si ψ se satisface entonces *a* es obligatoria"

4 Especificación de Procesos

Existen al menos tres semánticas posibles que pueden ser asociadas a una especificación de proceso en OASIS:

(a) La semántica del lenguaje utilizado para su especificación. En OASIS se usa la noción de *sistema de transición etiquetado* como se propone en CCS.

(b) La semántica de prohibiciones, secuencia de acciones permitidas.

(c) La semántica de obligaciones, secuencia de acciones que deben ocurrir.

Desde el punto de vista de la expresividad del lenguaje, la semántica del formalismo utilizado para especificar procesos, podría aplicarse a todo tipo de proceso, sin distinción. Esta será la semántica elemental que asumiremos por defecto para cada especificación de proceso.

Adicionalmente asociaremos la semántica (b) ó (c) a un proceso. Ésta es declarada de forma explícita. Así, la sección *operations* se utiliza para especificar **procesos de obligación**, la sección *patterns* para especificar **procesos de prohibición**. Un caso particular de proceso de obligación es el que actúa como una transacción, es decir, con la política del "todo o nada". En este caso se podría declarar con una calificativo de **transaction** para el proceso, asumiendo por defecto que el proceso no actúa como transacción.

Para especificar procesos se utiliza un cálculo de procesos, subconjunto del lenguaje de CCS, el cual se bosqueja a continuación.

Existe una estrecha relación entre el concepto de agente en CCS y el concepto de proceso en OASIS. Siguiendo el planteamiento y notación propuestos en CCS, la especificación de un proceso P puede ser vista como un sistema de transiciones etiquetado representado por (κ, *Act*, $\{\xrightarrow{a} : a \in Act\}$), donde:

- κ es el conjunto constantes agente que definen el proceso.
- *Act* es el conjunto de acciones usadas en la especificación del proceso, $Act \subseteq A$ es el conjunto de acciones en signatura de la clase.
- $\xrightarrow{a} \subseteq \kappa \times \kappa$ es la relación de transición

El conjunto de expresiones que definen cada constante agente se denota por ε y está formado al menos por el conjunto de constantes agente κ. Sea $E \in \kappa$, las expresiones agentes que definen a cada constante agente pueden ser formadas como se indica a continuación[1]:

(1) a.E , un Prefijo ($a \in Act$)
(2) $\sum_{i \in I} E_i$, una Suma (I es un *indexing set*)
(3) **if** C **then** E , una Condicional, C una expresión *boolean* definida sobre el estado del objeto.

Adicionalmente, **if** b **then** E_1 **else** E_2 puede ser definida como

(**if** C **then** E_1) + (**if** $\neg C$ **then** E_2)

Para cada constante agente E se tiene una ecuación de definición $E =_{def} Q$, siendo Q una expresión agente usando las operadores presentados. Así, un agente (proceso) es especificado mediante un conjunto de ecuaciones, una para cada constante agente. Por ejemplo, el proceso B está especificado por:

$B_1 =_{def} a.B_2 + a.B_3$
$B_2 =_{def} b.B_4$
$B_3 =_{def} c.B_5$
$B_4 =_{def} \mathbf{0}$
$B_5 =_{def} d.B_1$

0 es una constante especial definida como $\mathbf{0} = \sum_{i \in \varnothing} E_i$, como en este caso $I = \varnothing$, desde **0** no existen transiciones.

[1] En la definición del lenguaje en CCS, la relación de transición es $\xrightarrow{a} \subseteq \varepsilon \times \varepsilon$. Así, al definir el lenguaje, $E \in \varepsilon$. En nuestro caso la relación de transición es $\xrightarrow{a} \subseteq \kappa \times \kappa$ con lo cual $E \in \kappa$. Como veremos, esto nos asegura que cada nodo del grafo de transiciones asociado al proceso se corresponde con una constante agente que define a dicho proceso.

El sistema de transiciones para el proceso B es:

$\kappa = \{B_1, B_2, B_3, B_4, B_5\}$
Act = {a,b,c,d}
$\xrightarrow{a} = \{(B_1,B_2), (B_1,B_3)\}$
$\xrightarrow{b} = \{(B_2,B_4)\}$
$\xrightarrow{c} = \{(B_3,B_5)\}$
$\xrightarrow{d} = \{(B_5,B_1)\}$

Un proceso puede ser representado por un grafo de transiciones. La Figura 4.1 muestra el grafo de transiciones para el proceso B antes especificado.

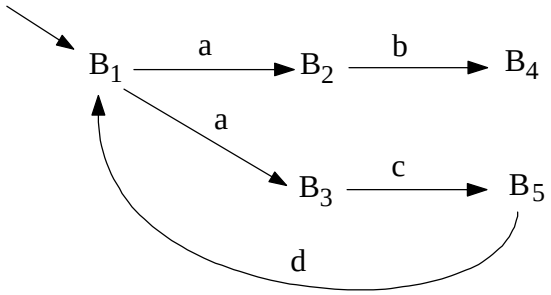

Figura 4.1

Llamaremos *estado del proceso* a la constante agente que en un determinado instante representa al proceso. De acuerdo al lenguaje definido para especificar procesos, un estado de proceso coincide con una constante agente del proceso y a su vez se corresponde con un nodo del grafo de transiciones del proceso.

Los estados que incluya una especificación de proceso corresponden a una decisión de modelamiento. Por otra parte, limitar que el dominio y codominio de la función de transición sean estados del proceso (coincidiendo con constantes agente del proceso) no disminuye la capacidad expresiva. Veamos el siguiente ejemplo para una especificación de proceso asociada a un cajero automático (ATM) simplificado:

```
ATM1 = new_ATM. ATM2;
ATM2 = destroy_ATM.ATM5 +
          read_card.(cancel + check_card_w_bank(n,p).ATM3).eject.ready. ATM2;
ATM3 = bad_PIN_msg + card_accepted. (cancel + issue_transaction(n,m).ATM4);
ATM4 = dispense_cash(m) + TA_failed_msg;
ATM5 = 0;
```

La Figura 4.2 muestra el grafo de transiciones para esta especificación de proceso. En esta especificación de proceso se han utilizado expresiones agente para definir a cada constante agente. Sin embargo, siempre es prosible obtener una especificación de proceso equivalente en la cual no se utilicen espresiones agente sino que sólo constantes agentes, coincidiendo éstas con los estados de proceso representados en el grafo de transiciones.

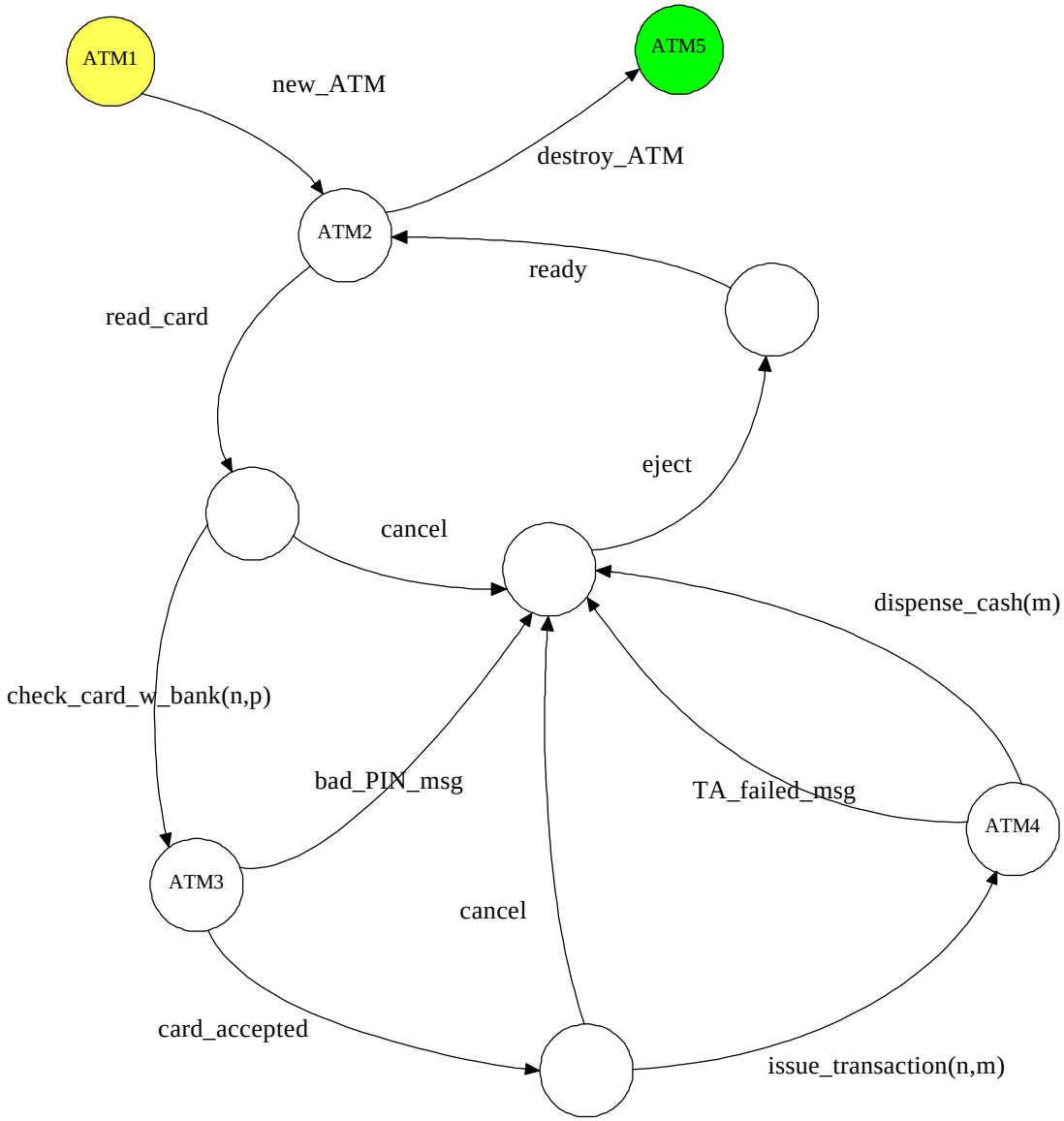

Figura 4.2

A continuación se presenta la especificación de proceso reescrita. La Figura 4.3 muestra el grafo de transiciones, el cual coincide con el anterior.

```
ATM1 = new_ATM. ATM2;
ATM2 = destroy_ATM.ATM9 + read_card. ATM3;
ATM3 = cancel.ATM7 + check_card_w_bank(n,p).ATM4;
ATM4 = bad_PIN_msg.ATM7 + card_accepted.ATM5;
ATM5 = cancel.ATM7 + issue_transaction(n,m).ATM6;
ATM6 = dispense_cash(m).ATM7 + TA_failed_msg.eject.ready.ATM7;
ATM7 = eject.ATM8;
ATM8 = ready.ATM2;
ATM9 = 0;
```

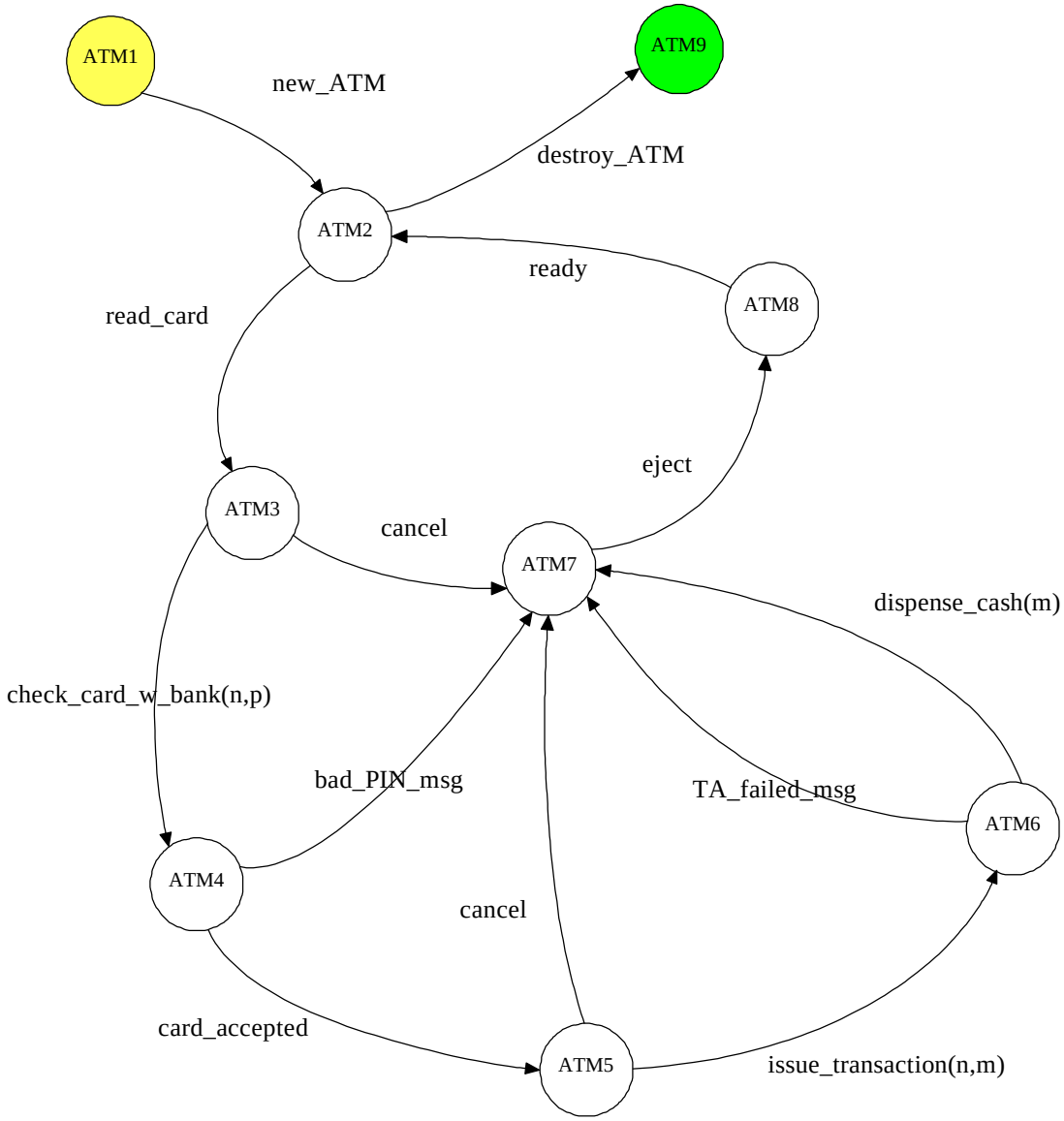

Figura 4.3

5 Procesos vistos como axiomas en Lógica Dinámica

Como se ha expuesto, los operadores de la Lógica Deóntica que expresan obligaciones, prohibiciones y permisos pueden ser vistos como una variante de Lógica Dinámica. Esto sugiere que los procesos de obligación y prohibición se corresponden del mismo modo con axiomas en Lógica Dinámica.

Una especificación de proceso determina una conducta particular del objeto. Diremos que *un proceso afecta a un objeto* si el objeto debe comportarse según la conducta establecida por dicho proceso. Comúnmente, el objeto seguirá la conducta del proceso en partes de su vida. Un caso particular es un proceso que afecta al objeto durante toda su existencia.

Un proceso de prohibición se puede expresar como un conjunto de axiomas en Lógica Dinámica de la forma $\neg\phi \rightarrow [a]$ *false*, del mismo modo como se hace para la sección *preconditions*.

De forma análoga, un proceso de obligación se puede expresar como un conjunto de axiomas en Lógica Dinámica de la forma $\phi \rightarrow [\neg a]$ *false*, del mismo modo como se hace para la sección *triggers*.

Como veremos, en ambos casos adicionalmente se obtendrán axiomas por transición de estado del proceso. Estos axiomas son del mismo tipo usado para evaluaciones, es decir, $\phi \rightarrow [a]\ \phi'$.

Sea P un proceso definido por el sistema de transiciones $(\kappa, Act, \{\xrightarrow{a} : a \in Act\})$, $E,E' \in \kappa$. Si $(E,E') \in \xrightarrow{a}$, esto suele escribirse como $E \xrightarrow{a} E'$ y se dice que *a es una acción de* E.

En la plantilla de clase OASIS, para cada proceso existe implícitamente un atributo variable de *sort* κ con el mismo nombre del proceso. Dicho atributo indica la constante agente (estado o nodo del grafo de transiciones asociado) en el cual se encuentra el proceso. La constante agente implícita **0** además se utiliza para indicar que el objeto no está siendo afectado por el proceso, es decir, si el proceso está en el nodo **0** en su sistema de transición, dicho proceso no está afectando el comportamiento del objeto.

Sea P un proceso y p el atributo variable asociado a su estado. Si *a es una acción de* E, llamaremos *predicado de ocurrencia de a en* E al predicado $(p=E \wedge C)$ y lo denotaremos por Ω_{Ea}. Siendo *C* la expresión *boolean* cuando E está definida por un operador condicional, sino el *átomo de ocurrencia* de *a* en E es sólo (p=E).

Ampliando dicho concepto al proceso, llamaremos *predicado de ocurrencia de a en* P, denotado por Ω_{Pa}, a la <u>disyunción</u> de *predicados de ocurrencia de a en* E, para cada constante agente E que define al proceso.

El significado de Ω_{Pa} es el siguiente: "Ω_{Pa} se satisface si y sólo si el proceso P está en algún estado E y un Ω_{Ea} se satisface, es decir, *a* es una ocurrencia de E y además se cumple la condición C asociada (si ésta existe)".

Así, el proceso P se corresponde con los siguientes axiomas en Lógica Dinámica:

Axiomas por transiciones

Efectúan la evolución del proceso de acuerdo a las transiciones definidas y a la ocurrencia de acciones.

Para cada $a \in Act$, y para cada $(E,E') \in \xrightarrow{a}$ se obtiene el siguiente axioma:

$$\Omega_{Ea} \rightarrow [a]\ p=E'$$

Siendo a_{new} la acción de creación para instancias de la clase. Si P es un proceso de obligación además existirá el axioma:

$$[a_{new}]\ p=\mathbf{0}$$

Si P es un proceso de prohibición el axioma adicional es:

$$[a_{new}]\ p=E_1$$

$E_1 \in \kappa$ es la constante agente de inicio del proceso.

Un proceso de obligación no afecta al objeto desde el comienzo de su existencia, un proceso de prohibición sí.

Axiomas de Prohibición (sólo si es un proceso de prohibición)

De acuerdo al estado del proceso, un axioma de prohibición establece la situación en la cual la ocurrencia de una acción se prohibe.

Para cada $a \in Act$ se obtiene el siguiente axioma:

$$\neg\Omega_{Pa} \rightarrow [a]\ false$$

Axiomas de Obligación (sólo si es un proceso de obligación)

De acuerdo al estado del proceso, un axioma de obligación establece la situación en la cual la ocurrencia de una acción es obligatoria.

Para cada $a \in Act$ se obtiene el siguiente axioma:

$$\Omega_{Pa} \rightarrow [\,\overline{a}\,]\ false$$

De esta forma, un proceso de obligación establece un tipo de comportamiento reactivo basado en secuencias de acciones (*event-driven*).

6 Plantilla de clase OASIS representada como una Teoría en Lógica Dinámica

Después de lo presentado, expresar la plantilla de una clase OASIS como una Teoría en Lógica Dinámica es inmediato. Sea $<Att, E, \Phi, Processes>$ la plantilla de una clase. Asociado al conjunto *E* de eventos, se tiene un conjunto *A* de acciones.

La teoría en Lógica Dinámica para una clase tiene los siguientes axiomas:

i. Para cada $a \in A$, y siendo ϕ la conjunción de todas las fórmulas de restricciones de integridad, axiomas del tipo:

$[a]\ \phi$

ii. Para cada $a \in A$ y para cada fórmula de derivación ($\phi \rightarrow \phi'$), un axioma del tipo:

$[a]\ (\phi \rightarrow \phi')$

iii. Para cada evaluación un axioma del tipo:

$\phi \rightarrow [a]\ \phi'$

iv. Para cada precondición un axioma de prohibición:

$\neg\phi \rightarrow [a]\ \mathit{false}$

v. Para cada disparo un axioma de obligación:

$\phi \rightarrow [\neg a]\ \mathit{false}$

vi. Para cada proceso de obligación P

Axiomas por transiciones

$[a_{new}]\ p=\mathbf{0}$

Para cada $(E,E') \in \xrightarrow{a}$

$\Omega_{Ea} \rightarrow [a]\ p=E'$

Axiomas de obligación

Para cada $a \in Act$

$\neg\Omega_{Pa} \rightarrow [a]\ \mathit{false}$

vii. Para cada proceso de prohibición P

Axiomas de por transiciones

$[a_{new}]\ p= E_1$, $E_1 \in \kappa$ es la constante agente de inicio del proceso

Para cada $(E,E') \in \xrightarrow{a}$

$\Omega_{Ea} \rightarrow [a]\ p=E'$

Axiomas de prohibición

Para cada $a \in Act$

$\Omega_{Pa} \rightarrow [\ \overline{a}\ ]\ false$

6.1 Ejemplo: Especificación de la clase Máquina de Chocolatinas

Se presenta una versión del clásico problema de la máquina de chocolatinas (vending machine) propuesto por Hoare [Hoa85]. Nuestra VM mostrada en la Figura 6.1 tiene las siguientes características:

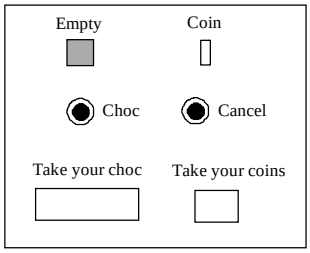

Figura 6.1

- Recibe sólo un tipo de monedas.
- Por cada moneda es posible obtener a cambio una chocolatina.
- La chocolatina es entregada cuando se presiona el botón "choc". Pueden introducirse hasta tres monedas antes de pedir una chocolatina. Por cada moneda introducida se almacena un crédito.
- Si el crédito es mayor que cero y no se desean chocolatinas pueden recuperarse las monedas presionando el botón "cancel".
- Cuando las chocolatinas se agotan, se enciende una luz indicando dicha situación. Sin embargo, pueden ser introducidas monedas. En este caso, el botón "choc" estará bloqueado.
- No se considera la recarga de chocolatinas para la máquina.

```
class vm
identification
        number: (number);
constant_attributes
        number: nat;
variable_attributes
        num_chocs: nat;
        credit: nat;
private_events
        set  new;
        coin_in;
        coin_out;
        choc;
        light_empty;
operations
        CANCEL.
                CANCEL1= {credit>1} ::coin_out.CANCEL1 +  {credit<=1}  ::coin_out;
valuations
        [coin_in] credit=credit+1;
        [coin_out] credit=credit-1;
        [choc] nchocs=nchocs-1  and  credit=credit-1;
        [::light_empty] switch_on=true;
preconditions
        choc   if   credit>0  and  nchocs>0;
        CANCEL  if   credit>0;
```

triggers
::light_empty **if** nchocs=0 **and** switch_on=false;
patterns
GETCHOC.
$GETCHOC_1$ = coin_in.$GETCHOC_2$;
$GETCHOC_2$ = choc.$GETCHOC_1$ + coin_in.$GETCHOC_3$ + coin_out.$GETCHOC_1$;
$GETCHOC_3$ = choc.$GETCHOC_2$ + coin_in.$GETCHOC_4$ + coin_out.$GETCHOC_2$;
$GETCHOC_4$ = choc.$GETCHOC_3$ + coin_out.$GETCHOC_3$;
end_class

Comentarios a la especificación del ejemplo:

a) En los capítulos posteriores se trata con detenimiento la sintaxis y expresividad de cada una de las secciones en la plantilla de una clase.

b) En la sintaxis utilizada, las acciones enviadas son precedidas por "::". Si antes de los "::" no se especifica el servidor, se asume que es **self**, es decir, el servidor es el mismo objeto. En el siguiente capítulo se describe en profundidad la representación de acciones.

c) CANCEL es un servicio de mayor nivel ofrecido por el objeto. Desde el punto de vista del cliente de dicho servicio, CANCEL es un servicio más, el cual corresponde al botón *cancel*. CANCEL es un proceso que obliga a que se ejecute una cierta secuencia de eventos. CANCEL establece que la máquina devuelva la cantidad de monedas asociada al crédito acumulado. La especificación de CANCEL es una simplificación de la notación:

$CANCEL_1$ = **if** {credit>1} **then** ::coin_out.$CANCEL_1$ **else** ::coin_out. $CANCEL_2$
$CANCEL_2$ = **0**

d) GETCHOC es un proceso que establece las secuencias válidas de ocurrencias de los eventos coin_in y choc.

e) En el ejemplo, los atributos variables implícitos serán getchoc y cancel. Estos atributos son de *sort* κ (constantes agente) y su valor está asociado al estado actual del proceso en la vida del objeto. Por lo tanto, si por ejemplo cancel=**0** el objeto no está bajo la influencia del proceso CANCEL.

Axiomas en Lógica Dinámica correspondientes al proceso CANCEL

CANCEL es un proceso con la semántica de obligación. El sistema de transición etiquetado asociado al proceso CANCEL es:

$\kappa = \{CANCEL_1\}$
Act = {::coin_out}
$\xrightarrow{coin\_out} = \{(CANCEL_1, CANCEL_1), (CANCEL_1, CANCEL_2)\}$

La Figura 6.2 muestra el grafo de transiciones correspondiente

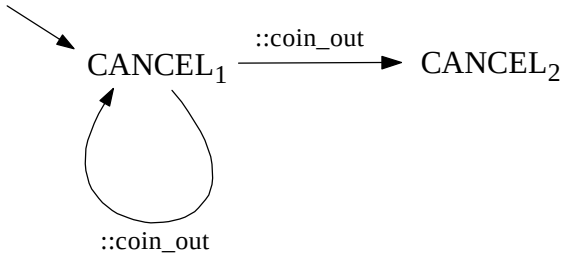

Figura 6.2

axiomas por transiciones

[set] cancel=**0**
[CANCEL] cancel=$CANCEL_1$
credit>1 $\wedge$ cancel=$CANCEL_1$ $\rightarrow$ [::coin_out] cancel=$CANCEL_1$
credit<=1 $\wedge$ cancel=$CANCEL_1$ $\rightarrow$ [::coin_out] cancel=**0**

axiomas de obligación[2]

(cancel=$CANCEL_1$ $\wedge$ credit>1) $\vee$ (cancel=$CANCEL_1$ $\wedge$ credit<=1) $\rightarrow$ [:: $\overline{\text{coin\_out}}$] *false*

Axiomas en Lógica Dinámica correspondientes al proceso GETCHOC

GETCHOC es un proceso con la semántica de prohibición. El siguiente sistema de transición etiquetado lo representa:

κ = {$GETCHOC_1$, $GETCHOC_2$, $GETCHOC_3$, $GETCHOC_4$}
Act = {coin_in, choc, coin_out}
$\xrightarrow{\text{coin\_in}}$ = {($GETCHOC_1$,$GETCHOC_2$), ($GETCHOC_2$,$GETCHOC_3$), ($GETCHOC_3$,$GETCHOC_4$)}
$\xrightarrow{\text{choc}}$ = {($GETCHOC_2$,$GETCHOC_1$), ($GETCHOC_3$,$GETCHOC_2$), ($GETCHOC_2$,$GETCHOC_3$)}
$\xrightarrow{\text{coin\_out}}$ = {($GETCHOC_2$,$GETCHOC_1$), ($GETCHOC_3$,$GETCHOC_2$), ($GETCHOC_2$,$GETCHOC_3$)}

La Figure 6.3 muestra el grafo de transiciones correspondiente al proceso GETCHOC.

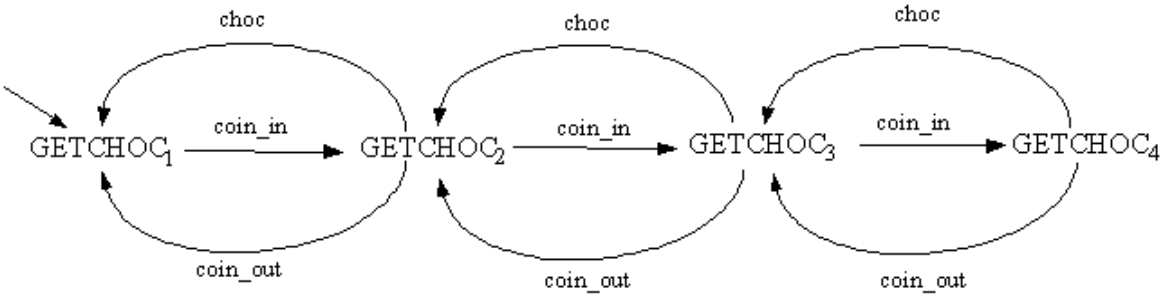

Figura 6.3

[2] El predicado de ocurrencia está tal cual se ha definido. Obviamente podrían hacerse simplificaciones.

axiomas por transiciones

$true \rightarrow$ [set] getchoc=GETCHOC$_1$
getchoc=GETCHOC$_1$ $\rightarrow$ [coin_in] getchoc=GETCHOC$_2$
getchoc=GETCHOC$_2$ $\rightarrow$ [coin_in] getchoc=GETCHOC$_3$
getchoc=GETCHOC$_3$ $\rightarrow$ [coin_in] getchoc=GETCHOC$_4$
getchoc=GETCHOC$_2$ $\rightarrow$ [choc] getchoc=GETCHOC$_1$
getchoc=GETCHOC$_3$ $\rightarrow$ [choc] getchoc=GETCHOC$_2$
getchoc=GETCHOC$_4$ $\rightarrow$ [choc] getchoc=GETCHOC$_3$
getchoc=GETCHOC$_2$ $\rightarrow$ [coin_out] getchoc=GETCHOC$_1$
getchoc=GETCHOC$_3$ $\rightarrow$ [coin_out] getchoc=GETCHOC$_2$
getchoc=GETCHOC$_4$ $\rightarrow$ [coin_out] getchoc=GETCHOC$_3$

axiomas de prohibición

$\neg$((getchoc=GETCHOC$_1$) $\vee$ (getchoc=GETCHOC$_2$) $\vee$ (getchoc=GETCHOC$_3$))$\rightarrow$[coin_in] *false*
$\neg$((getchoc=GETCHOC$_2$) $\vee$ (getchoc=GETCHOC$_3$) $\vee$ (getchoc=GETCHOC$_4$))$\rightarrow$ [choc] *false*
$\neg$((getchoc=GETCHOC$_2$) $\vee$ (getchoc=GETCHOC$_3$) $\vee$ (getchoc=GETCHOC$_4$))$\rightarrow$ [coin_out] *false*

Teoría Lógica Dinámica asociada a la clase vm

La especificación de la clase vm es una Teoría en Lógica Dinámica representada por los siguientes axiomas:

axiomas correspondientes a la sección *valuations*

credit=N $\rightarrow$ [coin_in] credit=N+1
credit=N $\rightarrow$ [coin_out] credit=N-1
nchocs=N $\wedge$ credit=M $\rightarrow$ [choc] nchocs=N-1 $\wedge$ credit=M-1
[::light_empty] switch_on=true

axiomas correspondientes a las transiciones del proceso GETCHOC

[set] getchoc=GETCHOC$_1$
getchoc=GETCHOC$_1$ $\rightarrow$ [coin_in] getchoc=GETCHOC$_2$
getchoc=GETCHOC$_2$ $\rightarrow$ [coin_in] getchoc=GETCHOC$_3$
getchoc=GETCHOC$_3$ $\rightarrow$ [coin_in] getchoc=GETCHOC$_4$
getchoc=GETCHOC$_2$ $\rightarrow$ [choc] getchoc=GETCHOC$_1$
getchoc=GETCHOC$_3$ $\rightarrow$ [choc] getchoc=GETCHOC$_2$
getchoc=GETCHOC$_4$ $\rightarrow$ [choc] getchoc=GETCHOC$_3$
getchoc=GETCHOC$_2$ $\rightarrow$ [coin_out] getchoc=GETCHOC$_1$
getchoc=GETCHOC$_3$ $\rightarrow$ [coin_out] getchoc=GETCHOC$_2$
getchoc=GETCHOC$_4$ $\rightarrow$ [coin_out] getchoc=GETCHOC$_3$

axiomas correspondientes a las transiciones del proceso CANCEL

[set] cancel=**0**
[CANCEL] cancel=CANCEL$_1$
cancel=CANCEL$_1$ $\wedge$ credit>1$\rightarrow$ [::coin_out] cancel=CANCEL$_1$
cancel=CANCEL$_1$ $\wedge$ credit=1$\rightarrow$ [::coin_out] cancel=**0**

axiomas correspondientes a la sección *preconditions*

$\neg$(nchocs>0 $\wedge$ credit>0) $\rightarrow$ [choc] *false*
$\neg$(credit>0) $\rightarrow$ [CANCEL] *false*

axiomas correspondientes a las prohibiciones en el proceso GETCHOC

$\neg((\text{getchoc}=\text{GETCHOC}_1) \vee (\text{getchoc}=\text{GETCHOC}_2) \vee (\text{getchoc}=\text{GETCHOC}_3)) \rightarrow$[coin_in] *false*
$\neg((\text{getchoc}=\text{GETCHOC}_2) \vee (\text{getchoc}=\text{GETCHOC}_3) \vee (\text{getchoc}=\text{GETCHOC}_4)) \rightarrow$ [choc] *false*
$\neg((\text{getchoc}=\text{GETCHOC}_2) \vee (\text{getchoc}=\text{GETCHOC}_3) \vee (\text{getchoc}=\text{GETCHOC}_4)) \rightarrow$ [coin_out] *false*

axioma correspondientes a la sección *triggers*

nchocs=0 $\wedge$ switch_on=false $\rightarrow$ [:: $\overline{\text{light\_empty}}$] *false*

axioma correspondientes a las obligaciones en el proceso CANCEL

(cancel=CANCEL_1 $\wedge$ credit>1) $\vee$ (cancel=CANCEL_1 $\wedge$ credit<=1) $\rightarrow$ [:: $\overline{\text{coin\_out}}$] *false*

La teoría en Lógica Dinámica que representa a cada instancia de la clase, estará formada por dos conjuntos de axiomas: los axiomas "fijos" (si no consideramos evolución del esquema) de la clase y los axiomas "cambiantes", que representan el estado del objeto en un instante dado.

```
class vm
identification
      number: (number);
constant_attributes
      number: nat;
variable_attributes
      num_chocs: nat;
      credit: nat;
      # atributos variables usados para cambios en los procesos #
      cancel:nat;
      getchoc:nat;
private_events
      set  new;
      coin_in;
      coin_out;
      choc;
      light_empty;
      # la operacion CANCEL es tratada como un evento #
          CANCEL;
valuations
      [coin_in] credit=credit+1;
      [coin_out] credit=credit-1;
      [choc] nchocs=nchocs-1  and  credit=credit-1;
         [::light_empty] switch_on=true
      # evaluaciones correspondientes a la operación CANCEL #
          [set] cancel=0;
      [CANCEL] cancel=1;
      credit>1 and cancel=1 [coin_out] cancel=1;
      credit<=1 and cancel=1 [coin_out] cancel=0;

      # evaluaciones correspondientes al pattern GETCHOC  #
```

```
      [set] getchoc=1;
      getchoc=1 [coin_in] getchoc=2;
      getchoc=2 [coin_in] getchos=3;
      getchoc=3 [coin_in] getchoc=4;
      getchoc=2 [choc] getchoc=1;
      getchoc=3 [choc] getchoc=2;
      getchoc=4 [choc] getchoc=3;
      getchoc=2 [coin_out] getchoc=1;
      getchoc=3 [coin_out] getchoc=2;
      getchoc=4 [coin_out] getchoc=3;
preconditions
      choc  if  credit>0  and  nchocs>0;
      CANCEL  if  credit>0;
      #  precondiciones correspondientes al pattern GETCHOC #
      coin_in  if  getchoc=1  or  getchoc=2  or getchoc=3;
      choc  if  getchoc=2 or getchoc=3  or getchoc=4;
      coin_out  if  getchoc=2 or getchoc=3  or getchoc=4;
triggers
         ::light_empty  if  nchocs=0 and  ∧ switch_on=false;
         # disparo correspondientes a la operación CANCEL #
         ::coin_out  if  (credir>1 and cancel=1) or (credir<=1 and cancel=1);
end_class
```

7 Ciclo de Vida de un Objeto

Llamaremos **estado** de un objeto a la conjunción de átomos del tipo *att=v*, donde *att* es un atributo del objeto y *v* su valor (elemento del *sort* de evaluación del atributo). Usaremos el estado del objeto para caracterizarlo en un instante determinado. Σ denota el conjunto de los estados alcanzables por un objeto y $\sigma \in \Sigma$ un estado en particular. El estado de un objeto cambia por la ocurrencia de acciones[3]. Los cambios de estado son modelados por la siguiente función:

$$eff: A \rightarrow (\Sigma \rightarrow \Sigma) \qquad (2.1)$$

eff es el cambio de estado atómico que produce la ocurrencia de una acción. En un mismo instante de tiempo a un objeto le puede acontecer más de una acción. Esta concurrencia intra-objetual es necesaria por dos motivos fundamentales, y sólo podría ser evitada a costa de unas fuertes restricciones en la expresividad del lenguaje. Las razones para considerar concurrencia intra-objetual son:

- En un mismo instante un objeto puede tener más de un disparo por realizar. Dichos disparos, por constituir obligaciones, deben ser ejecutados en el mismo instante en el cual se satisfacen sus condiciones.

- OASIS incluye la modelización de agregaciones, es decir, objetos compuestos por otros objetos. En este caso, debe existir una sincronización de las vidas de los componentes respecto del agregado. Sin perder generalidad, si un objeto agregado es visto como alguna forma de composición de objetos, en un instante determinado de la vida del

[3] Utilizaremos la *frame assumption*, es decir, supondremos que si no se especifica que la modificación del valor de un atributo se debe a la ocurrencia de una determinada acción, entonces dicho atributo no es modificado por dicha acción.

objeto agregado puede ocurrir más de un evento (algunos de ellos simplemente por sincronización con eventos de sus componentes).

Un conjunto $S \subseteq A$ es **consistente** si a partir de un estado σ dado, el estado σ' alcanzado por la ocurrencia de las acciones en S es siempre el mismo, independiente del orden en el cual se ejecuten las acciones.

Llamaremos **paso**[4] a un conjunto consistente de acciones (términos de acción en su forma *ground*), $\underline{\mu}_i \subseteq \underline{A}$ con i=0,1,... , que ocurren en un mismo instante de la vida del objeto.

Sea $\mu=\{a_1, ..., a_n\} \subseteq A$ un paso. La relación de accesibilidad entre estados para μ, $R_\mu \subseteq \Sigma \times \Sigma$ es definida como [Wie93][Dig96]:

$$R_\mu(\sigma,\sigma') \Leftrightarrow_{def} \; eff(a_n)(...(\, eff'(a_1)...)(\sigma) = \sigma' \qquad (2.2)$$

De acuerdo a la definición de un conjunto consistente, el orden en el cual se aplica la función *eff* no altera el estado resultante del objeto.

Llamaremos **vida** o **traza** de un objeto a la secuencia de pasos que le acontecen a lo largo de su existencia.

8 Semántica de una Especificación OASIS

La semántica de OASIS es dada en términos de una estructura de Kripke (W, τ, ρ). W es el conjunto de todos los mundos posibles que un objeto puede alcanzar. Sea F el conjunto de F.b.f. en Lógica de Predicados de Primer Orden evaluadas sobre el estado (mundo) en el cual se encuentra el objeto, A el conjunto de acciones en la signatura del objeto y $M \subseteq 2^A$, el conjunto de pasos posibles. Las funciones τ y ρ se definen como:

$$\tau : F \rightarrow 2^W \qquad (2.3)$$
$$\rho : M \rightarrow (W \rightarrow W) \qquad (2.4)$$

La función τ asigna a una fórmula en Lógica de Predicados de Primer Orden el conjunto de mundos en los cuales se satisface. La función ρ asigna a cada paso una relación binaria entre mundos, la cual es la semántica declarativa del lenguaje. Sea $\mu \in M$ y $w,w' \in W$, el significado buscado es: $(w,w') \in \rho(\mu)$ si y sólo si la ocurrencia[5] de μ conduce al objeto desde el mundo w al mundo w'. Cada par (w,w') obtenido según lo anterior se llama **transición válida**.

La relación entre estados, mundos, pasos y la vida del objeto se representa en la Figura 8.1. t_i es el tiempo en el cual se produce el i-ésimo *tic* de reloj para el objeto.

MUNDOS	inexistencia	→	w_1	→	w_2	→	...	→	w_i	→	...	→ destrucción
ESTADOS			σ_1		σ_2		...		σ_i		...	
PASOS	$\underline{\mu}_0$		$\underline{\mu}_1$		$\underline{\mu}_2$		...		$\underline{\mu}_i$		...	

[4] En [Jun95] este concepto recibe el nombre de *snapshot*. En [Wie93] se denomina *step*.

[5] Usaremos indistintamente las expresiones "ocurrencia" y "ejecución" para referirnos a una acción o paso que acontece en la vida del objeto.

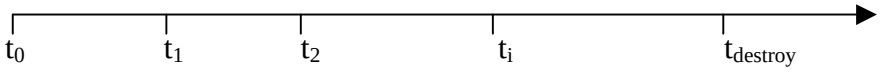

Figura 8.1

Así, la interpretación de los diferentes axiomas en Lógica Dinámica usados en una plantilla de clase OASIS es la siguiente:

a) Axiomas del tipo $\psi \rightarrow [a]\ \phi$, en particular $[a]\ \phi$, es decir, cuando ψ es **true** (cualquier estado del objeto).

   Sea $w \in \tau(\psi)$ el mundo actual, si *a* ocurre entonces el mundo alcanzado es $w' \in \tau(\phi)$. Así, las evaluaciones establecen las propiedades que deben satisfacerse en el mundo alcanzado como efecto atómico de la ejecución de una acción. Si el efecto producido por un paso μ esta representado por $R_\mu(\sigma,\sigma')$, y con $\{a\} \subseteq \mu$, entonces se cumple que $\sigma \models \psi$ y $\sigma' \models \phi$.

b) Axiomas de prohibición $\neg\phi \rightarrow [a]\ false$

   Sea $w \in \tau(\neg\phi)$ el mundo actual, si *a* ocurre, no existe por definición una transición válida. Es decir, un axioma de prohibición impide que una acción ocurra estando el objeto en ciertos estados, aquellos donde $\sigma \models \neg\phi$.

c) Axiomas de obligación $\phi \rightarrow [\neg a]\ false$

   Sea $w \in \tau(\phi)$ el mundo actual, si la acción *a* no ocurre, no existe por definición una transición válida. Es decir, una obligación fuerza a que la acción *a* ocurra cada vez que el estado del objeto es tal que $\sigma \models \phi$.

9 Conclusiones y Trabajo Futuro

A partir de las nociones de phohibición y obligación (proporcionadas por una variante de Lógica Dinámica, extendida con operadores deónticos) se ha propuesto un marco formal uniforme para las diferentes secciones de una plantilla de clase OASIS.

Se ha redefinido y ampliado el uso de especificaciones de proceso, introduciendo el concepto de *pattern* y de *operation*, que vienen a sustituir los processes y transactions de OASIS 2.2.

Operations son procesos de obligación con la idea “todo o nada” (transacciones) o sin ella.
Patterns son procesos de prohibición

Una clase puede tener más de un proceso en *operations* y/o *patterns*. Es más sencillo especificar las vidas posibles como la composición de subprocesos. Cada proceso utiliza en su definición un subconjunto de las acciones en la signatura de la clase. Se ha presentado las correspondencias de las especificaciones de proceso utilizadas con fórmulas en Lógica Dinámica. Así, su semántica es obtenido dentro del marco homogéneo de Lógica Dinámica al igual que para el resto de secciones de la clase.

El poder representar una clase como una teoría en Lógica Dinámica abre un camino muy interesante hacia la verificación de consistencia y análisis estático de especificaciones OASIS.

Cualquier proceso podría afectar sólo a partes de la vida del objeto. Por otra parte, más de un proceso puede en un mismo instante estar afectando el comportamiento del objeto.

El modelo de ejecución planteado para animación especificaciones OASIS [Let97] ya incluye el tratamiento de prohibiciones y obligaciones, con lo cual según lo planteado en este trabajo, la animación de especificaciones de proceso se obtiene en forma inmediata.

Referencias

[Agh86] Agha G.A. ACTORS: A model of Concurrent Computation in Distributed Systems. The MIT Press, 1986.

[Åqv84] Åqvist L. Deontic logic. In D.M. Gabbay and F.Guenthner, editors, Hnadbook of Philosophical Logic II, pp.605-714. Reidel, 1984.

[Dig96] Dignum,F., Meyer J.-J.Ch., Wieringa R.J., Kuiper R.A. Modal Approach to Intentions, Commitments and Obligations: Intention plus Commitment yields Obligation., Springer 1996.

[Dub93] Dubois,E., Du Bois P., Petit M. O-O Requirements Analysis: an agent perspective. In Proc. of the 7th.European Conference on Object Oriented Programming-ECOOP 93, pp. 458-481. July 1993.

[Fee93] Feenstra R.B., Wieringa R.J. LCM 3.0: a languaje for describing conceptual models. Tchnical Report IR-344, Faculty of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, December 1993.

[Gen94] Genesereth M.R., Ketchpel S.P. Software Agents. Communications of the ACM, 37(7):48-53, 1994.

[Har84] Harel,D. Dynamic Logic in Handbook of Philosophical Logic II, editors D.M.Gabbay, F.Guenthner; pags. 497-694. Reidel 1984.

[Hoa85] Hoare, C.A.R. Communicating Sequential Processes. Prentice Hall, 1985.

[Jun95] Jungclaus R., Saake G., Hartmann T., Sernadas C.TROLL - A Language for Object-Oriented Specification of Information Systems ACM Transactions on Information Systems, Volume 14, Number 2, Pages 175-211, April 1995.

[Let97] Letelier P., Sánchez P., Ramos I. Animación de Modelos Conceptuales para ayudar en la Validación de Requisitos, ASOO´97, Buenos Aires, Argentina, Agosto 1997.

[Mey88] Meyer J.-J.Ch. A different approach to deontic logic: Deontic logic viewed as a variant of dynamic logic. In Notre Dame Journal of Formal Logic, vol.29, pages 109-136, 1988.

[Mil89] Milner R. Communication and Concurrency. Prentice Hall Series in Computer Science. C.A.R. Hoare, Series Editor. 1989.

[Pas92] Pastor O. Diseño y Desarrollo de un Entorno de Producción Automática de Sotfware basado en el modelo OO, Tesis Doctoral, DSIC-UPV, Mayo 1992.

[Pas95] 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, Servicio de Publicaciones Universidad Politécnica de Valencia, 1995.

[Pas97] Pastor O., Insfrán E., Pelechano V.,Romero J.,Merseguer J. "OO-METHOD: An OO Software Production Environment Combining Conventional and Formal Methods". Conference on Advanced Information Systems Engineering (CAiSE '97). Barcelona, Spain. June 1997.

[Ram93] Ramos I., Pastor O., Cuevas J., Devesa J. Objects as Observable Processes, Actas del 3rd. Workshop on the Deductive Approach to Information System Design, Costa Brava (Cataluña), 1993.

[Ser92] Sernadas C., Gouveia P., Sernadas A. Oblog: Object-Oriented, logic-based conceptual modelling. Research Report, Instituto Superior Técnico, Secção de Ciêcia da Computação, Departamento de Matemática, 1096 Lisboa, Portugal, 1992.

[Ser94] Sernadas C., Ramos J. Gnome: Sintaxe, semântica e cálculo. Research Report, Instituto Superior Técnico, Secção de Ciêcia da Computação, Departamento de Matemática, 1096 Lisboa, Portugal, 1994.

[Wie92] Wieringa R.J. A conceptual model specification language (CMSL Version 2). 1992.

[Wie93] 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.