Un ambiente para especificación incremental y validación de modelos conceptuales*

Patricio Letelier **Pedro Sánchez** **Isidro Ramos**

Departamento Sistemas Informáticos y Computación

Universidad Politécnica de Valencia

Camino de Vera s/n, 46071 Valencia

email {letelier | ppalma | iramos}@dsic.upv.es

http://www.dsic.upv.es/users/oom

Resumen

Las dificultades encontradas en la captura, especificación, verificación y validación de requisitos han impulsado variadas propuestas que mejoran dicho proceso. El modelo conceptual establece los requisitos funcionales del software y es uno de los resultados principales de dichas actividades, constituyéndose en una pieza clave para las posteriores tareas en el desarrollo de software. Los métodos formales dotan de precisión y rigor al modelo conceptual, pero presentan dificultades en cuanto a su aplicación y comprensión por el usuario. Por otro lado, las técnicas de prototipación han ganado aceptación por su fácil aplicación y proximidad al usuario, pero en general carecen de exhaustividad y precisión. Actualmente, la animación de modelos conceptuales formales emerge como un enfoque capaz de aprovechar las ventajas de los métodos formales y de la prototipación. OASIS es un enfoque formal para el modelado conceptual OO. El comportamiento de un objeto en OASIS está determinado por un conjunto de fórmulas expresadas mediante una variante de lógica dinámica. Este trabajo presenta un ambiente integrado para el modelado conceptual basado en OASIS, orientado especialmente a la especificación incremental y validación mediante animación.

1 Introducción

El modelo conceptual, representando los requisitos funcionales de un sistema de información, es la pieza clave para establecer el vínculo entre el espacio del problema y el espacio de la solución. Las deficiencias del modelo conceptual tienen un impacto considerable en las posteriores actividades del proceso de desarrollo de software.

La construcción del modelo conceptual es un proceso de descubrimiento, no sólo para el analista, sino también para el usuario. La estrategia que más se ajusta a esta situación es construir el modelo conceptual de forma iterativa e incremental, mediante interacción entre el usuario y el analista. Durante el modelado conceptual se llevan a cabo esencialmente cuatro tareas: captura de requisitos, modelado o especificación, verificación de criterios de calidad y consistencia, y finalmente validación. Una vez alcanzada la conformidad del analista y del usuario, el modelo conceptual es utilizado como entrada para las fases posteriores del desarrollo. El grado de exhaustividad en las validaciones con el usuario es una variable determinante para conseguir que el producto final se ajuste a lo esperado. La necesidad de validación del modelo conceptual está presente cada vez que se efectúa alguna modificación en él, sea ésta durante su proceso incremental de construcción, durante el diseño e implementación o durante el mantenimiento.

Los métodos formales [27] para el desarrollo de software (y en particular para el modelado conceptual) prometen una mejora en la calidad del resultado pues su aplicación permite expresar y determinar con rigor las propiedades del software. Sin embargo, la introducción de métodos formales no ha sido inmediata y sólo en la actualidad comienzan tímidamente a ser utilizados en el entorno industrial de

*Este trabajo ha sido financiado por el proyecto MENHIR de la Comisión Interministerial de Ciencia y Tecnología, con referencia TIC97-0593-C05-01.

desarrollo de software [2, 25]. Cuatro razones explican esta situación [9]: falsas expectativas o desconocimiento, carencia de estándares para métodos y su aplicación, falta de herramientas integradas en entornos CASE y finalmente una precaria preparación entre sus usuarios potenciales.

En particular, para validar los requisitos del sistema se ha extendido la utilización de técnicas de prototipación. Un prototipo permite al usuario y al analista determinar las propiedades del sistema sobre una representación más cómoda para el usuario. Generalmente los prototipos son bosquejos de la interfaz del producto, concentrándose principalmente en aspectos de entrada y salida del sistema. Indudablemente la prototipación es una ayuda para la validación del modelo conceptual, pero no es suficiente. Un tipo particular de prototipación está orientada a simulación o animación del sistema. Este tipo de prototipación resulta atractiva para cubrir de forma exploratoria la validación del modelo conceptual, analizando su comportamiento ante situaciones de funcionamiento más complejas. Sin embargo, en esta estrategia se presentan dos importantes obstáculos: conseguir una representación ejecutable del modelo conceptual e interactuar con el usuario usando dicha representación.

De acuerdo a lo anteriormente expuesto y centrándose en las dificultades presentes en la construcción de modelos conceptuales, la utilización de enfoques formales aporta mejoras principalmente en aspectos de rigor y precisión de la especificación, facilitando su verificación. Sin embargo, respecto de la captura y validación de requisitos, las técnicas de prototipación han ganado más aceptación. De esto se deduce que los métodos formales y las técnicas de prototipación son complementarios, por lo cual resulta interesante una combinación de ambos. En los últimos años se ha registrado gran interés en torno a la validación de especificaciones formales mediante su animación [24]. En este sentido, este trabajo propone una enfoque para la animación de especificaciones formales mediante la ejecución de la especificación y orientada a la exploración interactiva del comportamiento del modelo conceptual y su validación respecto de escenarios [21].

Se utilizará OASIS 3.0 (**O**pen and **A**ctive **S**pecification of **I**nformation **S**ystems) [12] como enfoque formal para la construcción de modelos conceptuales. Otras propuestas con una motivación similar a la de OASIS son: TROLL [10], LCM [5], Albert [4] y Oblog [23]. Algunas propuestas relacionadas con validación mediante animación en dichos trabajos son: la presentada en [8] que utiliza TROLL, y en [7] basada en Albert. Respecto de la animación establecida, las diferencias entre otras propuestas y este trabajo radican principalmente en las características formales del enfoque utilizado y la expresividad ofrecida. Además, considerando los resultados alcanzados, el estado del arte es similar y se limita a versiones preliminares de entornos para animación.

El asegurar la fidelidad de la animación respecto del modelo conceptual presenta los mismos problemas que se tienen en la fase de pruebas para verificar la corrección de la implementación respecto del modelo conceptual. Sin embargo, en este trabajo dicho problema es atenuado por los siguientes factores: a) el modelo conceptual está basado en un enfoque formal y se ha establecido un modelo abstracto de ejecución [16] que está inspirado en la semántica de dicho enfoque formal y b) la animación se implementa ([13, 15, 22]) en entornos de programación concurrentes que ofrecen la posibilidad de justificar formalmente las equivalencias entre constructores de la especificación OASIS del sistema y el programa obtenido para animación.

En este trabajo se presenta un ambiente para especificación incremental y validación de requisitos basado en OASIS. Se describen las características generales del entorno que, como es de esperar, se enmarcan dentro de las funcionalidades normalmente provistas por los entornos CASE actuales, sin embargo, en este trabajo se pone énfasis en aspectos de validación lo cual constituye la excepción y el aporte más significativo.

En el apartado 2 se exponen brevemente los aspectos básicos de OASIS. El apartado 3 presenta una propuesta para la construcción de un ambiente para especificación incremental y validación de requisitos. Los conceptos básicos del modelo de ejecución utilizado para animar especificaciones OASIS son introducidos en el apartado 4. El entorno gráfico para animación de especificaciones OASIS es descrito en el apartado 5. Finalmente, se presentan las conclusiones en el apartado 6.

2 OASIS

OASIS [12] es un enfoque formal para la especificación de modelos conceptuales usando el paradigma orientado a objeto. En OASIS un sistema de información es visto como una sociedad de objetos autónomos y concurrentes que se comunican entre sí a través de acciones.

OASIS está formalizado a través de una variante de Lógica Dinámica [6] que permite representar

los operadores de obligación, prohibición y permiso usados en Lógica Deóntica [1]. Una especificación OASIS es una presentación de una teoría en el sistema formal usado, expresada como un conjunto estructurado de definiciones de clase. Las clases pueden ser simples o complejas. Una clase compleja es aquella que en su definición utiliza la definición de otras clases (simples o complejas). Las clases complejas se definen estableciendo relaciones entre clases. Estas son agregación y herencia.

Una clase se compone de un nombre de clase, uno o más mecanismos de identificación para instancias de la clase (objetos) y un tipo o plantilla que comparten todas las instancias. La plantilla recoge las propiedades de estructura y de comportamiento compartidas por todos los objetos potenciales de la clase considerada. Para una clase simple todas las propiedades quedan establecidas en su plantilla. Para una clase compleja, además de las propiedades establecidas por su plantilla, existirán propiedades adicionales determinadas por los operadores de clase que la definen. A continuación se presentan los conceptos básicos de OASIS.

Definición 1 ***Plantilla o tipo.*** *Una plantilla de clase está representada por una tupla* $\langle Atributos, Eventos, Fórmulas, Procesos \rangle$.

- *Atributos* es el alfabeto de atributos. $\forall att \in Atributos$, se tiene la función att : *sort de nombrado* $\rightarrow$ *sort de evaluación*
- *Eventos* es el alfabeto genérico de eventos. $\forall e \in Eventos$ podemos obtener $\underline{e} = \theta e$ siendo θ una substitución básica de los posibles parámetros del evento.
- *Fórmulas* es un conjunto de fórmulas en diversas lógicas según sea la sección del lenguaje donde se utilizan.
- *Procesos* es el conjunto de especificaciones de proceso. Cada proceso se especificará usando un cálculo de procesos basado en un subconjunto del lenguaje propuesto en CCS [18]. Las especificaciones de proceso serán divididas en operaciones y protocolos.

Definición 2 ***Servicio.*** *Un servicio es un evento o una operación. En el primer caso se trata de un servicio atómico e instantáneo. Una operación es un servicio no atómico y que, en general, tiene duración.*

Definición 3 ***Acción.*** *Una acción es una tupla* $\langle Cliente, Servidor, Servicio \rangle$*, que representa tanto la acción asociada a requerir el servicio en el objeto cliente como la acción asociada a proveer el servicio en el objeto servidor.*

Para cada clase, asumiremos la existencia implícita de un conjunto A de acciones obtenido a partir de los servicios que los objetos de la clase pueden requerir (cuando son clientes) o proveer (cuando son servidores). 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 servicio.

Definición 4 ***Atributo.*** *Un atributo es un par* $\langle$*nombre, sort de evaluación*$\rangle$*, siendo nombre el identificador del atributo y sort de evaluación el nombre del conjunto soporte (carrier) con los valores que puede tomar dicho atributo.*

Definición 5 ***Estado del objeto.*** *Llamaremos estado del objeto al conjunto de sus atributos evaluados. Se expresa normalmente mediante Fórmulas bien formadas (Fbfs) de una lógica de primer orden.*

El estado permite caracterizar a un objeto en un instante determinado. Sea Σ el conjunto de estados alcanzables por un objeto y $\sigma \in \Sigma$ un estado en particular. El estado de un objeto cambia por la ocurrencia de acciones. Los cambios de estado son modelados por la siguiente relación:

$$efecto : \underline{A} \rightarrow (\Sigma \rightarrow \Sigma)$$

Donde $\underline{A}$ es el conjunto de acciones instanciadas ($\underline{a} \in \underline{A}$) que le pueden acontecer al objeto y $efecto$ es la función que asocia a cada acción instanciada una función de cambio de estado, siendo $(\Sigma \rightarrow \Sigma)$ el conjunto de todas las funciones de cambio de estado.

Definición 6 ***Conjunto consistente de acciones.*** *Un conjunto de acciones* $M \subseteq \underline{A}$ *es consistente si a partir de un estado* σ *dado, el estado* σ' *alcanzado por la ocurrencia de las acciones en* M *es siempre el mismo, independiente del orden en el cual se ejecuten*[1] *las acciones.*

Definición 7 ***Paso.*** *Llamaremos paso a un conjunto consistente de acciones que ocurren en un mismo instante de la vida del objeto.*

[1] Se utilizará "ejecución de acción" como sinónimo de "ocurrencia de acción" para hacer más intuitiva la presentación. Sin embargo, desde el punto de vista declarativo y formal, "ocurrencia" es el término más adecuado.

Sea $\mu = \{a_1, ..., a_n\} \subseteq \underline{A}$ un paso. La relación de accesibilidad entre estados está determinada por μ, $R_\mu \subseteq \Sigma \times \Sigma$ y se define como (siendo "$\circ$" el operador de composición funcional):

$$R_\mu(\sigma) \stackrel{def}{\equiv} efecto(a_1)(\sigma) \circ ... \circ efecto(a_n)(\sigma) = \sigma'$$

De acuerdo con la definición de conjunto consistente, el orden en el cual se aplica la función $efecto$ a las acciones no altera el estado siguiente alcanzado por el objeto.

Definición 8 ***Vida o traza de un objeto.*** *Llamaremos vida o traza de un objeto a un prefijo finito de pasos que le acontecen al objeto. De esta forma, la traza es una caracterización alternativa del estado del objeto en un instante determinado.*

2.1 OASIS expresado en Lógica Dinámica

Globalmente y considerando su expresividad, OASIS está basado en Lógica Deóntica. La Lógica Deóntica es la lógica de obligaciones, prohibiciones y permisos. Siendo $a \in A$, en OASIS se utilizan los siguientes operadores de la Lógica Deóntica:

$O(a)$	obligación de ocurrencia de a.
$F(a)$	prohibición de ocurrencia de a.
$P(a)$	permiso o habilitación de ocurrencia de a.

En [17] la Lógica Deóntica es descrita como una variante de Lógica Dinámica. La definición de los operadores deónticos en Lógica Dinámica es la siguiente:

$\psi \rightarrow [a]false$	"la ocurrencia de a está prohibida en los estados que satisfacen ψ".
$\psi \rightarrow [\neg a]false$	"la ocurrencia de a es obligatoria en los estados que satisfacen ψ".
$\psi \rightarrow [a]\phi$	"en los estados que satisfacen ψ, inmediatamente después de la ocurrencia de la acción a, ϕ debe satisfacerse".

Donde ψ es una Fbf que caracteriza el estado del objeto inmediatamente antes de la ocurrencia de la acción a y $\neg a$ representa la no-ocurrencia de la acción a (es decir, que no ocurre nada o que ocurre otra acción distinta de a). Además, $false$ es un átomo tal que no hay un estado que pueda hacerlo verdadero. Esta situación podría representar un estado de violación del sistema. El significado intuitivo de éstas fórmulas es: una acción está prohibida si su ocurrencia conduce a un estado de violación. Una acción es obligatoria si su no-ocurrencia 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 como "después de la ocurrencia de la acción a, ϕ debe satisfacerse", siendo a una acción y ϕ una Fbf de la lógica de primer orden utilizada para caracterizar el estado. Estas fórmulas constituyen un sublenguaje del lenguaje de Lógica Dinámica propuesto y formalizado en [26].

En OASIS, el modelado del comportamiento puede realizarse desde dos perspectivas complementarias en cuanto a expresividad. En cada instante, para un objeto, las acciones permitidas y las acciones obligatorias pueden ser establecidas de la siguiente forma:

- Considerando el estado del objeto en dicho instante. Las precondiciones son fórmulas que permiten la ocurrencia de acciones sólo en ciertos estados del objeto. De forma análoga, los disparos son fórmulas que obligan la ocurrencia de acciones en determinados estados del objeto.

- Considerando las acciones ya acontecidas. Una especificación de proceso determina un patrón de comportamiento para el objeto desde el punto de vista de secuencias de pasos acontecidos, es decir, determina trazas posibles. En OASIS se utilizan dos tipos de especificaciones de proceso: *protocols* y *operations*. Un protocolo establece secuencias permitidas de acciones. Una operación establece secuencias obligatorias de acciones. En [14] se explica la interpretación de una especificación de proceso como un conjunto de fórmulas en la variante de lógica dinámica utilizada.

La semántica de OASIS es dada en términos de una estructura de Kripke (W, τ, ρ). W es el conjunto de todos los mundos[2] posibles que un objeto puede alcanzar.

[2] De acuerdo con lo dicho, los estados son aserciones (fórmulas), los mundos son estructuras sobre las que dichas fórmulas son interpretadas.

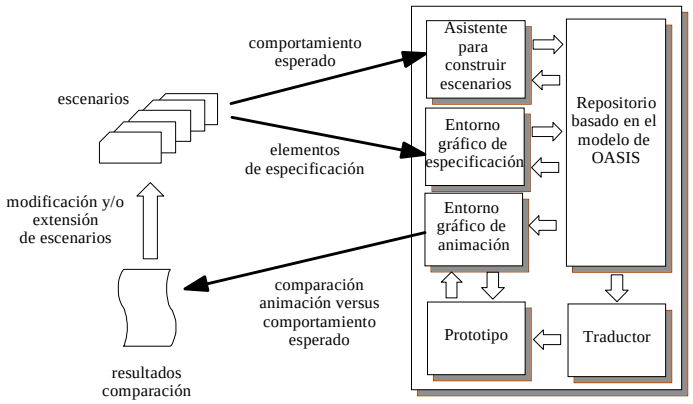

Figura 1: Un ambiente para construcción incremental y validación de modelos conceptuales

Sea F el conjunto de Fbfs evaluadas sobre el estado (mundo asociado) en el cual se encuentra el objeto, A el conjunto de acciones de la signatura del objeto y $2^{\underline{A}}$, el conjunto de pasos posibles instaciados. Las funciones τ y ρ se definen como:

$$\tau : F \rightarrow 2^W$$
$$\rho : 2^{\underline{A}} \rightarrow (W \rightarrow W)$$

La función τ asigna a una fórmula en la lógica de estado utilizada (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. Siendo $\mu \in 2^{\underline{A}}$ y $w, w' \in W$, el significado buscado es: $(w, w') \in \rho(\mu)$ si y sólo si la ocurrencia de μ conduce al objeto desde el mundo w al mundo w'.

3 Un ambiente para especificación y validación

El modelo conceptual debería ser construido incrementalmente mediante interacción entre el usuario y el analista. Como se trata de un proceso de descubrimiento, en cada paso sería recomendable que el modelo conceptual fuera validado respecto de los requisitos del usuario. Normalmente esta tarea de comparación se realiza manualmente y por simple inspección. Cualquier apoyo automatizado para el proceso de validación debe disponer de una representación para los requisitos que pueda contrastarse respecto de la funcionalidad del modelo.

La Figura 1 ilustra el ambiente para la especificación incremental y validación de modelos conceptuales que se está construyendo. En términos generales, se trata de un trabajo que puede ser extendido hacia el desarrollo de una herramienta CASE. Sin embargo, se ha prescindido de características no estrictamente relacionadas con la construcción incremental y validación del modelo conceptual. La programación de las interfaces gráficas se está realizando con Tcl/Tk [19]. Los componentes del entorno (en el rectángulo de la derecha en la Figura 1) son:

- Un asistente para construir escenarios en un formato adecuado para su comparación. La programación de este módulo aún no se ha comenzado. Las técnicas usadas para especificar escenarios varían considerablemente (ver[21]). Como el interés se centra en realizar una comparación de resultados obtenidos por animación respecto de resultados esperados incluidos en los escenarios, tanto la especificación de escenarios como la animación del modelo conceptual estarán caracterizadas por trazas de acciones y estados alcanzados de la vida de los objetos estudiados.

- Un entorno gráfico de especificación, que permita editar el modelo conceptual. Este módulo permite la edición gráfica del modelo, definiendo los elementos de la especificación que se almacenan en el repositorio. Se ha construido una versión preliminar que permite especificar gráficamente la configuración de clases y en forma textual los detalles de la plantilla de una clase. Alternativamente a la introducción textual, para las especificaciones de proceso de una clase (operaciones y protocolos) se está integrando un editor gráfico de diagramas

de transición de estados. La notación gráfica utilizada está adaptándose para hacerla compatible con UML [20].

- El repositorio, que almacena el modelo conceptual y los escenarios. Corresponde a un metamodelo estático para especificaciones OASIS visto como un modelo relacional (detallado en [11]).
- Un traductor, que genera automáticamente un prototipo a partir del modelo conceptual almacenado. Se ha construido un traductor que a partir del repositorio genera un programa lógico concurrente en KL1[3] cuyo ejecutable constituye el prototipo para la animación. El traductor implementa las pautas presentadas en [15].
- El entorno de animación, que interactúa con el prototipo validando cada uno de los escenarios respecto del comportamiento del prototipo. En el contexto de OASIS el comportamiento del sistema se corresponde con el comportamiento de la sociedad de objetos incluidos en él y puede ser observado a través de las acciones acontecidas y los estados alcanzados por cada objeto. Así, la animación de una especificación OASIS permite examinar el comportamiento de un objeto o grupo de objetos, estudiando las acciones acontecidas y los correspondientes estados alcanzados. Esta facilidad exploratoria constituye una mejora para la validación del modelo conceptual. Sin embargo, el aporte más significativo se conseguirá cuando se puedan contrastar con asistencia automática las trazas esperadas respecto de las obtenidas por animación. Se ha construido una versión preliminar para este entorno [15], sin incluir la funcionalidad de comparación de trazas. Posteriormente se explicará con más detalle este módulo.

El proceso de construcción del modelo conceptual, desde la perspectiva del ambiente propuesto, está compuesto por las siguientes actividades:

1. Capturar requisitos del usuario mediante técnicas de escenarios.
2. Especificar los escenarios en una forma representativa para su comparación (objetos, acciones y estados alcanzados).
3. Especificar los elementos de la especificación asociados a cada escenario y que determinan partes del modelo conceptual.
4. Verificación automática de la consistencia del modelo conceptual y de los escenarios introducidos.
5. Obtención automática de un prototipo para animación.
6. Validación del modelo conceptual. Mediante animación de la especificación se contrasta el funcionamiento del sistema respecto de cada escenario introducido. Además, realizando una animación exploratoria pueden buscarse situaciones de funcionamiento que evidencien comportamientos no deseados del modelo conceptual o que determinen ampliaciones.
7. Modificación y/o extensión del conjunto de escenarios.

Estas actividades se repiten en la secuencia establecida hasta conseguir un resultado satisfactorio en lo que respecta a las actividades 4 y 6. Además, las actividades 2 y 3 se realizan en conjunto por ser dependientes. Cualquier modificación en los requisitos funcionales del sistema que determine cambios en los escenarios podría activar nuevamente dicho ciclo de actividades.

4 Un modelo de ejecución para OASIS

Siendo la utilidad fundamental de este trabajo su uso en la validación de los requisitos expresados en una especificación OASIS, es indispensable mantener la implementación lo más cercana posible a la semántica de OASIS. De esta forma existirá un camino de formalización y verificación de la implementación obtenida respecto de la semántica de una especificación OASIS. En este sentido, el modelo de ejecución usado es un animador abstracto de fórmulas de obligación, permiso y cambio de estado, asociadas a un objeto. La información utilizada por el modelo es la signatura de atributos y acciones del objeto junto al conjunto de fórmulas en lógica dinámica que describen su comportamiento. A continuación se describen brevemente los conceptos incluidos en el modelo de ejecución propuesto en [16].

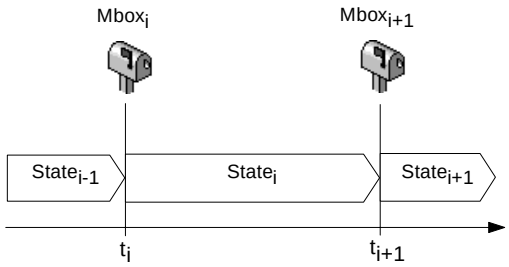

Figura 2: Ciclo de vida de un objeto

4.1 Aspectos básicos

La secuencia de pasos que le acontecen a un objeto está ordenada respecto del tiempo. Podemos suponer que existe un objeto "reloj" que envía acciones especiales — llamadas *ticks* — hacia cada objeto del sistema[3]. Los *ticks* recibidos por un objeto son correlativos con los números naturales, t_1, t_2, etc. Sean i y j números naturales tales que se cumple $i < j \Longleftrightarrow t_i < t_j$.

Definición 9 ***Buzón***. *Es el conjunto de acciones que pueden formar parte del paso que un objeto ejecuta en un determinado tick. El buzón en el instante t_i se denota por $Mbox_i$. Consideraremos cada buzón como un buffer ilimitado, en el que las acciones se ordenan por orden de llegada.*

Definición 10 ***Estado i-ésimo del objeto***. *Denotado por $State_i$, representa el estado del objeto en el intervalo $[t_i, t_{i+1})$. Es decir, desde t_i inclusive y hasta justo antes de t_{i+1}, el estado se considera constante.*

La Figura 2 ilustra la vida de un objeto según las definiciones dadas. El conjunto de acciones en cada paso es obtenido desde el buzón y la ejecución del paso en un instante produce un cambio de estado que no será observable hasta el siguiente *tick*. Así, en la estructura de tiempo utilizada aunque la ocurrencia de acciones es instantánea, los estados tienen una cierta duración.

4.2 Clasificación de acciones en el buzón

El procesamiento de las acciones del buzón implica clasificar las acciones contenidas en él. A continuación se presentan las categorías de acciones identificables en el buzón, caracterizadas como subconjuntos de $Mbox_i$ (es decir, en el instante t_i).

Definición 11 ***Acciones obligadas de requerir***. *Acciones asociadas a obligaciones por requerir un servicio (en las cuales el cliente es el propio objeto) y que* ***deben*** *acontecer. El conjunto de acciones obligadas de requerir se denota por $OReq_i$.*

Las acciones de requerir están determinadas por fórmulas de obligación, es decir, de la forma: $\phi[\neg a]false$, en las cuales a es la tupla $< self, Servidor, Servicio >$, es decir, el cliente de la acción es el propio objeto, indicado usando la palabra $self$.

Definición 12 ***Acciones obligadas de proveer***. *Acciones correspondientes a servicios que el objeto* ***debe*** *proveer. Estas acciones están asociadas a obligaciones, en las cuales el cliente no es el propio objeto. El conjunto de acciones obligadas de proveer en se denota por $OProv_i$.*

Las acciones obligadas de proveer están determinadas por fórmulas de obligación, es decir, de la forma: $\phi[\neg a]false$, en las cuales a es la tupla $< Cliente, self, Servicio >$, es decir, el servidor de la acción es el propio objeto.

Definición 13 ***Acciones obligadas de ejecutar***. *Este conjunto se denotado por $OExec_i$ y se define como: $OExec_i = OReq_i \cup OProv_i$.*

Supondremos que el garantizar la no-existencia de conflicto[4] entre acciones obligadas es parte de la verificación estática de la especificación. Es decir, en tiempo de ejecución no puede presentarse conflicto entre obligaciones. Del mismo modo, supondremos que no puede darse una situación en la cual una acción obligada no está permitida.

[3] Para propósitos de animación, en este punto no tiene mayor importancia el que el reloj sea sólo uno para todo el sistema, uno para cada objeto o cualquier otra alternativa.

[4] En el modelo de ejecución se establece que dos acciones están en conflicto si afectan a conjuntos no disjuntos de atributos.

Definición 14 ***Acciones no-obligadas de ejecutar.*** *Acciones correspondientes a servicios solicitados por otros objetos (o por el propio objeto) y que el objeto* ***puede*** *proveer, dependiendo de los permisos establecidos sobre dichas acciones y verificados sobre $State_i$. El conjunto de acciones no-obligadas se denota por $\overline{O}Exec_i$.*

Definición 15 ***Acciones rechazadas.*** *Es un subconjunto de $\overline{O}Exec_i$ constituido por acciones no-obligadas de ejecutar cuya ocurrencia no está permitida. El conjunto de acciones rechazadas se denota por $Reject_i$.*

Las acciones rechazadas están determinadas por fórmulas de prohibición, es decir, de la forma: $\phi[a]false$.

Definición 16 ***Acciones candidatas.*** *Es un subconjunto de $\overline{O}Exec_i$ constituido por acciones no-obligadas de ejecutar cuya ocurrencia está permitida. El conjunto de acciones candidatas se denota por $Cand_i$.*

Además, se cumple que $\overline{O}Exec_i = Reject_i \cup Cand_i$.

Definición 17 ***Acciones ejecutadas.*** *Es el conjunto de acciones correspondiente al paso que se ejecuta en t_i. El paso se denota por $Exec_i$. El paso estará constituido por $OExec_i$ más un subconjunto de $Cand_i$, es decir, acciones no-obligadas de ejecutar que están permitidas.*

La selección del subconjunto de acciones de $Cand_i$ que serán incluidas en $Exec_i$ puede responder a distintos criterios y la única restricción es que no se produzcan conflictos entre las acciones del paso.

Definición 18 ***Acciones en conflicto.*** *Es un subconjunto de $Cand_i$ constituido por acciones no-obligadas de ejecutar cuya ocurrencia está permitida, pero que están en conflicto con alguna de las acciones obligadas de ejecutar o con alguna de las acciones no-obligadas seleccionadas. El conjunto de acciones en conflicto se denota por $Conf_i$.*

Se utiliza un criterio sencillo para la selección de acciones desde $Cand_i$: ante dos acciones que no pueden ser seleccionadas por estar en conflicto se seleccionará aquella llegada antes al buzón. Este criterio evita postergación indefinida de acciones en conflicto. Las acciones en conflicto $Conf_i$ serán "copiadas" al buzón correspondiente al siguiente instante ($Mbox_{i+1}$).

El comportamiento del objeto está caracterizado por un algoritmo que repetidamente procesa el buzón para determinar los subconjuntos definidos anteriormente y produce el cambio de estado del objeto.

Las equivalencias establecidas entre conceptos de OASIS y construcciones en programación lógica concurrente [13] junto al modelo de ejecución han permitido definir las pautas generales para traducir una especificación OASIS hacia el lenguaje lógico concurrente KL1. Posteriormente, se ha construido un traductor que recupera la especificación almacenada en el repositorio, genera un programa KL1 y lo compila, obteniéndose finalmente un prototipo ejecutable de la especificación. El módulo asociado al entorno gráfico de animación permite al analista interactuar de una forma cómoda con el prototipo. La versión preliminar de este módulo incluye un análisis básico de trazas pero aún no incorpora la comparación con trazas esperadas obtenidas de los escenarios definidos. El siguiente apartado muestra la funcionalidad hasta ahora implementa en dicho módulo a través de un ejemplo.

5 Entorno de animación

El siguiente ejemplo en OASIS modela un sistema bancario de "juguete".

```
conceptual schema banking_system
class account
identification
    number:(number);
constant attributes
    number:nat;
    name:string;
variable attributes
    balance:nat(0);
    times:nat(0);
    pin:nat(0);
    rank:nat(0);
derived attributes
    good_balance:bool;
derivations
    good_balance:={balance>=100};
events
    open new;
    close destroy;
```

```
    deposit(Amount:nat);
    withdraw(Pin:nat,Amount:nat);
    pay_commission;
    change_pin(Pin:nat,NewPin:nat);
    change_rank(Rank:nat);
valuations
    [deposit(Amount)]
        balance:=balance+Amount,
        times:=times+1;
    [withdraw(Pin,Amount)]
        balance:=balance-Amount,
        times:=times+1;
    [self:pay_commission[5]]
        balance:=balance-1;
    [::pay_commission]
        times:=0;
    [change_pin(Pin,NewPin)]
        pin:=NewPin;
    [change_rank(Rank)]
        rank:=Rank;
preconditions
    withdraw(Pin,Amount) if
        (pin=Pin and balance>=Amount) or
            (pin=Pin and balance<Amount
        and rank=2);
    change_pin(Pin,NewPin) if (pin=Pin);
    close if (balance=0);
triggers
    ::pay_commission when
        (times>=5 and good_balance=false
        and rank=0);
end class

class customer
identification
    name:(name);
constant attributes
    name:string;
events
    add new;
    remove destroy;
end class

interface customer(someone)
            with account(someone)
    services(deposit,withdraw,change_pin);
end interface

interface account(someone) with self
    services(pay_commission);
end interface

end conceptual schema
```

La Figura 3 ilustra el inicio de la sesión de animación. La animación empieza desde el ambiente integrado, donde previamente se carga (o se crea) un esquema conceptual, se verifica su consistencia y se genera el prototipo. La ventana de la Figura 3 está dividida en dos áreas: a la izquierda el área de objetos permite visualizar los objetos que están activos y a la derecha se muestran las listas de acciones procesadas hasta el último *tick* significativo (aquel en el cual algún objeto alcanzó un nuevo estado). Se ha implementado un mecanismo versátil de filtros para la visualización de acciones y facilitar el análisis de las trazas de objetos. Por un lado, en el área de objetos se puede seleccionar un objeto, un grupo de objetos o todos los objetos del esquema conceptual (haciendo click en el fondo del área izquierda). Adicionalmente, con los botones del área derecha, se puede seleccionar el subconjunto de acciones que se desea visualizar. Por defecto se muestran todas ($Mbox_i$), pero se pueden filtrar las obligadas de requerir ($OReq_i$), las no-obligadas de ejecutar ($\overline{O}Exec_i$), las acciones en conflicto ($Conf_i$), las acciones ejecutadas ($Exec_i$) o las acciones rechazadas ($Reject_i$)[6]. Los botones de la parte inferior del área izquierda tienen la siguiente funcionalidad: el botón *play* comienza la animación; el botón *stop* detiene la animación para comenzar una nueva con *play*; el botón pausa suspende la animación y habilita los botones *rewind* y *forward* que permiten retroceder o avanzar examinando las trazas en *ticks* ya ejecutados (el área de la derecha se refresca a medida que se cambia el *tick*. Al avanzar o retroceder sólo se muestran los *ticks* significativos.

Al comienzo, se refleja la creación de un metaobjeto por cada clase del esquema. Además, también por defecto, se crea un objeto `user` identificado por `root`. A través de este objeto el analista puede crear las instancias necesarias de las clases correspondientes.

[5] En OASIS, para evitar detallar cada elemento de la tupla $< Cliente, Servidor, Servicio >$ que representa a una acción, se adoptan las siguientes simplicaciones: a) Si el servidor es el propio objeto, se escribe *Cliente:Servicio* y si el cliente no es relevante sólo se escribe *Servicio*. b) Si el cliente es el propio objeto, se escribe *Servidor::Servicio*. *c)* En el caso particular de comunicación *self*, la acción de requerir se escribe *::Servicio* y la acción de proveer *self:Servicio* (o simplemente *Servicio*).

[6] En esta versión del traductor y entorno de animación no ha sido incorporado el tratamiento de acciones obligadas de proveer ($OProv_i$).

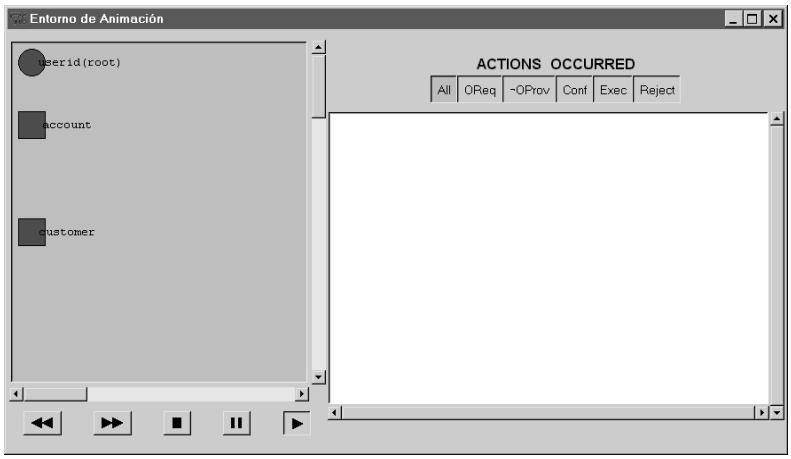

Figura 3: Inicio de la sesión de animación

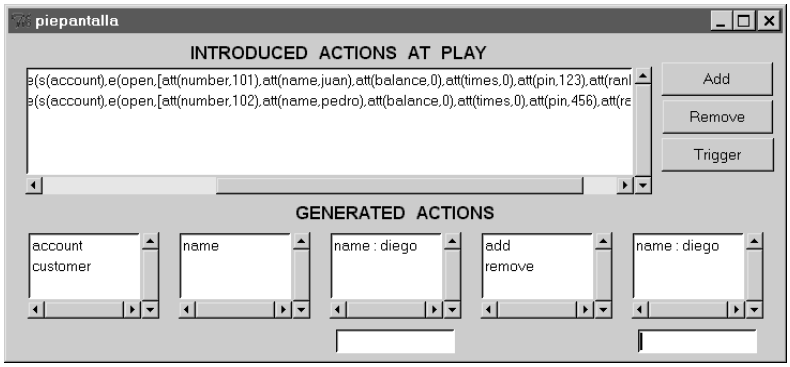

Figura 4: Construcción de acciones de creación de instancias

Al hacer un click sobre un objeto aparece la ventana de la Figura 4. Esta ventana muestra todas las clases y servicios que son accesibles para el objeto seleccionado. Además, se mostrará una ventana indicando el estado del objeto seleccionado. Por defecto, el objeto `user(root)` puede acceder a todos las clases del esquema y a todos sus servicios. Para otros objetos dicho acceso depende exclusivamente de las interfaces establecidas en el esquema conceptual. En la ventana de la Figura 4 se realiza la creación de dos instancias de cuenta (con número `101` y `102`) y una instancia de cliente identificado por el nombre "diego".

Después de lanzar las acciones de creación y hacer click en el fondo del área de objetos, la ventana resultante se muestra en la Figura 5. A continuación, se recreará un escenario en el cual un objeto cliente realiza cuatro depósitos y un reintegro en una cuenta. Los servicios solicitados son: `deposit(10)`, `deposit(20)`, `deposit(30)`, `deposit(40)` y `withdraw(50)`. Seleccionando el objeto `customer(diego)` se accederá a una ventana similar a la Figura 4 desde la cual el analista puede construir las acciones y requerir los servicios en representación del objeto `customer(diego)`. Las cinco acciones serán enviadas al objeto `account(101)`.

Si posteriormente se selecciona el objeto `account(101)` y se presiona el botón $Exec_i$, puede verse en la Figura 6 la lista de todas las acciones ejecutadas hasta el último *tick* significativo. La Figura 7 muestra el estado del objeto `account(101)` a partir de dicho *tick*.

Las limitaciones de presentación del presente trabajo impiden ilustrar más exhaustivamente una situación de exploración de trazas ejecutadas. Sólo es posible mencionar las principales funcionalidades del entorno tal como se ha intentado.

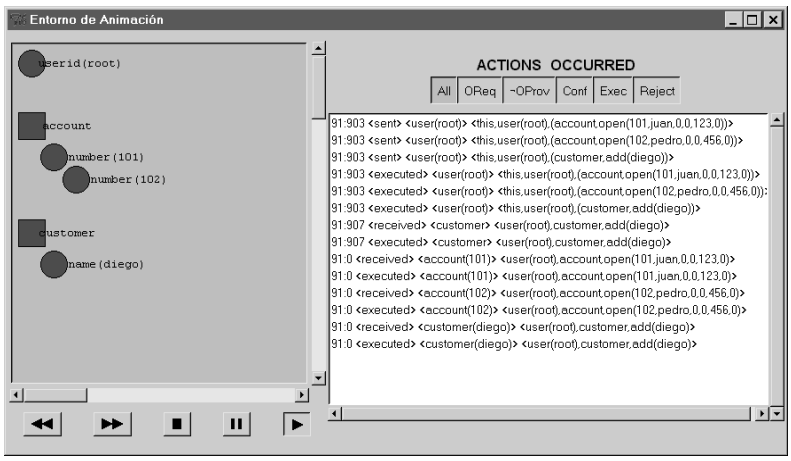

Figura 5: Sesión de animación después de crear tres instancias

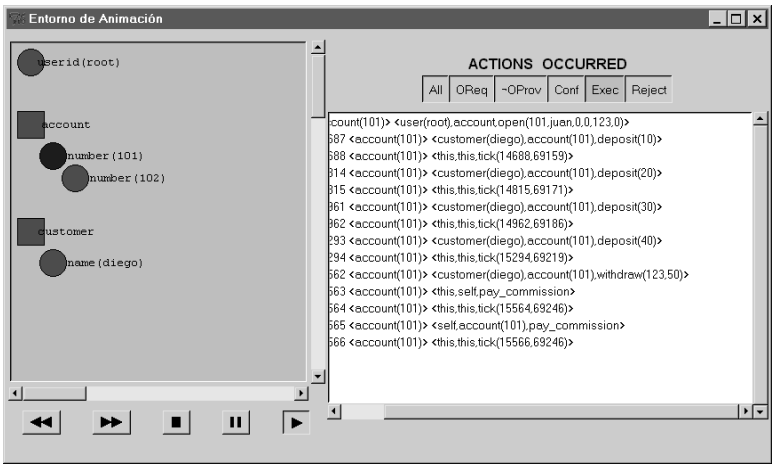

Figura 6: Acciones ejecutadas por el objeto `account(101)`

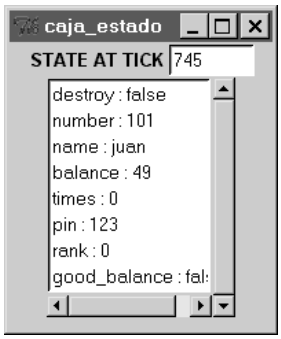

Figura 7: Estado actual del objeto `account(101)`

6 Conclusiones

En OASIS, el comportamiento de un objeto es formalizado con una variante de lógica dinámica y se expresa como un conjunto de fórmulas que representan obligaciones, permisos y cambios de estado. Así, el prototipo automáticamente obtenido es un animador para este tipos de fórmulas. El asegurar la fidelidad de la animación respecto del modelo conceptual OASIS presenta los mismos problemas que se tienen en la fase de pruebas para verificar la corrección de la implementación respecto del modelo conceptual. Sin embargo, en este trabajo dicho problema es atenuado por los siguientes factores: a) el modelo conceptual está basado en un enfoque formal, para el cual se ha establecido un modelo abstracto de ejecución inspirado en la semántica de OASIS y b) la animación que se ha propuesto está implementada en programación lógica concurrente, la cual ofrece la posibilidad de justificar formalmente las equivalencias entre constructores de la especificación OASIS del sistema y el programa obtenido para animación.

Se han expuesto las líneas generales de un ambiente integrado para la especificación de modelos conceptuales. Aunque este trabajo, se enmarca dentro del contexto de herramientas CASE, el énfasis se ha puesto en la validación temprana de requisitos mediante la animación de la especificación, usando un prototipo generado automáticamente. Esto ha hecho prescindir de funcionalidades que normalmente están presentes en un CASE pero que no tienen una relación estrecha con la especificación y validación del modelo conceptual. El ambiente propuesto está en construción y los módulos componentes se encuentran en distintos estados de avance. No se ha cubierto toda la expresividad de OASIS y las extensiones necesarias son aún considerables. Se ha bosquejado el proceso de modelado conceptual apoyado por animación. El enfoque utilizado, que combina un modelo formal con técnicas de escenarios y prototipación automática de especificaciones, provee mejoras significativas al proceso de modelado conceptual.

Referencias

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

[2] Craigen D., Gerhart S., Ralston T. Formal methods reality check: Industrial usage. IEEE Transactions on Software Engineering, vol.21, n.2, pp. 90-98, February 1995.

[3] Chikayama, T., KLIC User's Manual. Institute for New Generation Computer Technology, Tokyo JAPAN, March 1995.

[4] 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.

[5] Feenstra R., Wieringa R.J. LCM 3.0: A language for describing conceptual models. Technical Report IR-344, Faculty of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, December 1993.

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

[7] Heymans P. The Albert II Specification Animator. Technical Report CREWS 97-13, Cooperative Requirements Engineering with Scenarios, http://sunsite.informatik.rwth-aachen.de/CREWS/reports97.htm.

[8] Herzig R., Gogolla M. An animator for the object specification language TROLL light. Proc.Colloq.on Object-Orientation in Databases and Software Engineering, Montreal 1994.

[9] Hinchey M.G., Bowen J.P. To formalize or not to formalize?. En mesa redonda "An invitation to formal methods", IEEE Computer, pp.18-19, Abril 1996.

[10] 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, pp.175-211, April 1995.

[11] Letelier P., Ramos I. Un metamodelo estático para especificaciones OASIS y su implementación en una base de datos relacional. Informe técnico DSIC-II/19/96, DSIC-Universidad Politécnica de Valencia, 1996.

[12] Letelier P., Ramos I., Sánchez P., Pastor O. OASIS 3.0: Un enfoque formal para el modelado conceptual orientado a objeto. Servicio de Publicaciones Universidad Politécnica de Valencia, SPUPV-98.4011, 1998.

[13] Letelier P., Sánchez P., Ramos I. Animation of system specifications using concurrent logic programming. Symposium on Logical Approaches to Agent Modeling and Design, ESSLLI'97, Aix-en-Provence, France, August 1997.

[14] Letelier P., Sánchez P., Ramos I. Especificaciones de proceso para objetos y su representación en lógica dinámica. Actas de las IV Jornadas de Ingeniería del Software JIS'98, Murcia, Noviembre 1998.

[15] Letelier P., Sánchez P., Ramos I. Prototyping a requirements specification through an automatically generated concurrent logic program. G. Gupta (Ed.): Practical Aspects of Declarative Languages, Springer-Verlag, LNCS 1551, pp. 31-45, 1998.

[16] Letelier P., Ramos I., Sánchez P., Pastor O. Un modelo de ejecución para especificaciones OASIS 3.0. Actas de las III Jornadas de Trabajo MENHIR, Murcia, Noviembre 1998.

[17] 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, pp.109-136, 1988.

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

[19] Ousterhout J. Tcl and the Tk Toolkit. Addison-Wesley, 1994.

[20] Rational Software Corporation. UML notation guide. Versión 1.1, Septiembre 1997. http://www.rational.com/uml.

[21] Rolland C., Ben Achour C., Cauvet C., Ralyté J., Sutcliffe A., Maiden N.A.M., Jarke M., Haumer P., Pohl K., Dubois E., Heymans P. A Proposal for a Scenario Classification Framework, Technical Report CREWS 96-01, Cooperative Requirements Engineering with Scenarios, http://sunsite.informatik.rwth-aachen.de/CREWS/reports96.htm.

[22] Sánchez P., Letelier P., Ramos I. Constructs for Prototyping Information Systems using Object Petri Nets, Proc.of IEEE International Conference on System Man and Cybernetics, pp. 4260-4265, Orlando, USA, Octubre 1997.

[23] 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.

[24] Siddiqi J., Morrey I.C., Roast C.R., Ozcan M.B. Towards quality requirements via animated formal specifications. Annals of Software Engineering, n.3, 1997.

[25] 2RARE. 2 Real Applications for Requirements Engineering. Deliverable D12: "Final report on user's results" and Deliverable 13: "Final report on supplier's results", October 1996. http://www.info.fundp.ac.be/~phe/2rare.html.

[26] 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.

[27] Wing J.M. A specifier's introduction to formal methods. IEEE Computer, pp.8-24, September 1990.