Sobre la aplicación de los resultados de investigación a la docencia de Ingeniería del Software

Carsí J.A., Ramos I., Canós J.H.
Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
e-mail: {pcarsi, iramos, jhcanos}@dsic.upv.es

Resumen

En las universidades españolas se dedican grandes esfuerzos a la investigación en numerosas áreas. A veces, dichos esfuerzos suelen tener poca o nula trascendencia en el curricula de las asignaturas en las que los propios investigadores imparten docencia. En la unidad docente de Desarrollo de Software del Departamento de Sistemas Informáticos y Computación de la UPV intentamos que esto no sea así, y en diversas asignaturas se ha realizado el esfuerzo de aplicar los resultados de la investigación convirtiéndolos, bien en temas de teoría, bien en prácticas interesantes que suelen gustar a los alumnos por su novedad o interés. En el presente artículo se presentan los contenidos de la asignatura "Tecnología Software Avanzada", una asignatura optativa de quinto curso. En las clases teóricas se transmite a los alumnos la importancia de los métodos formales como una vía para el desarrollo de software de calidad, mientras que las prácticas están relacionadas con la utilización de esos métodos formales para el diseño e implementación de aplicaciones robustas con altas cotas de calidad utilizando procedimientos automáticos.

1. Introducción

Actualmente, en los departamentos universitarios españoles se dedican grandes esfuerzos a la investigación en numerosas áreas. En general, los temas de investigación son muy avanzados (por la propia naturaleza de la actividad investigadora), y los investigadores transmiten los conocimientos adquiridos en los programas de tercer ciclo. Es sabido que los índices de matriculación en estudios de doctorado son considerablemente menores que en estudios de pregrado, lo que hace que la difusión (en el ámbito docente) del saber adquirido sea relativamente pequeña si la comparamos con el esfuerzo realizado.

Dado que la difusión de los resultados es uno de los objetivos primordiales del trabajo universitario, pensamos que se debería tender a acercar más el trabajo docente y el investigador, extendiendo la difusión, en lo posible, al segundo ciclo (idealmente, al último curso de la carrera). La "distancia" entre investigación y docencia de primer y (sobre todo) segundo ciclo es mucho menor en disciplinas que podríamos considerar aplicadas, como la práctica de la Ingeniería, que en las puramente teóricas, sencillamente porque no es tanto el bagaje conceptual con el que se debe contar para llegar a comprender hasta qué punto una investigación ha producido frutos interesantes.

Dentro del mundo de la Informática, uno de los campos en que mayor trascendencia puede llegar a tener la transferencia de conocimientos es el de la Ingeniería del Software, en concreto toda aquella investigación relacionada con la definición de métodos, el uso y/o la construcción de herramientas de desarrollo.

Con este espíritu, en la unidad docente se ha realizado el esfuerzo de intentar aplicar los resultados de la investigación a ciertas asignaturas de último curso de Ingeniería en Informática, convirtiéndolos, bien en temas de teoría, bien en prácticas, con resultados francamente satisfactorios. De hecho, ya en los últimos años se han empleado en clases de laboratorio herramientas producidas en el transcurso de proyectos de investigación en Ingeniería de Requisitos por parte del Grupo de Modelado y Bases de Datos Orientadas a Objetos, y por supuesto se han dedicado clases teóricas para presentar sus fundamentos al nivel apropiado para el uso que de ellas se iba a hacer.

Buen ejemplo de la aplicación de la investigación a la docencia es la asignatura *Tecnología Software Avanzada* (TSA), que se presenta en este artículo fundamentalmente en su vertiente práctica. En TSA se muestra como automatizar la implementación a partir de especificaciones de alto nivel de los sistemas haciendo uso de entornos de producción de software industriales actuales.

2. OASIS: Un Lenguaje de Especificación Formal de Sistemas de Información

El principal punto del temario de teoría de la asignatura de TSA es la presentación de OASIS [4] como modelo de objetos de referencia, con todo lujo de detalles, para posteriormente compararlo con otros modelo como el ODMG-93 y CORBA.

OASIS es un lenguaje de especificación formal de sistemas de información abiertos y activos utilizado en [2] como lenguaje único para el diseño, consulta y manipulación de BDOO. En OASIS, los bloques básicos de construcción de los sistemas de información son los **objetos**. Un objeto es un proceso observable que encapsula estado y comportamiento. El estado está caracterizado por el conjunto de valores de las propiedades estáticas de los objetos, llamadas **atributos**. Un objeto tiene un identificador (oid), que es independiente del estado del objeto y permanece inalterado siempre. Por otra parte, el comportamiento de los objetos se define por los **servicios** (eventos y transacciones) que ofrece al resto de la sociedad de objetos.

En OASIS se utiliza la lógica para representar el estado de los objetos; un objeto en un estado dado es una teoría en alguna lógica; fbf caracterizan el valor de los atributos y existen reglas que definen el valor de los **atributos derivados** (atributos cuyo valor depende del valor de otros atributos). Un conjunto de fórmulas representa la forma en que cambian los valores de los atributos debido a la ocurrencia de los eventos. En [2], se usa la lógica dinámica para representar dichas fórmulas. Otras aproximaciones se han seguido usando diferentes lógicas como, por ejemplo, la *Transaction Frame Logic* ([1]).

Otra parte del comportamiento de los objetos está relacionada con los conceptos deónticos de prohibición y obligación. Por un lado, la **precondición** de un evento hace que su ocurrencia no sea relevante. Por otro lado, las **relaciones de disparo** permiten definir el comportamiento activo de los objetos en función de su estado.

Objetos similares (que comparten las mismas propiedades o tipo) son dinámicamente agrupados en **clases**. La especificación de una clase es la descripción de las propiedades que una colección de objetos[1] (la extensión de la clase) comparte. Las clases proporcionan mecanismos para la creación y destrucción de objetos, así como facilidades para la generación de oids.

La funcionalidad de las clases se formaliza a través de la noción de **meta-objeto**. Las clases son objetos con un conjunto de propiedades observables y un conjunto de servicios que manipulan dichas propiedades. En el modelo OASIS todo objeto es instancia de alguna clase, la clase de la cual son instancias las clases es la metaclase OASIS. Los servicios que ofrecen las clases incluyen los eventos de creación y destrucción de sus instancias. Entre los servicios que ofrecen las clases se encuentra la población de la clase.

La complejidad de la descripción del mundo real requiere distinguir entre diferentes tipos de clases: **clases primitivas** (que representan los tipos abstractos de datos), **clases elementales** y **clases complejas** (construidas a partir de otras clases utilizando operadores entre clases tales como la especialización, la agregación y otros).

El **proceso** representado por un objeto es construido sobre el alfabeto de eventos usando operadores tomados de la teoría de procesos. Operadores como la elección, la secuencia..., son la parte funcional del álgebra de procesos usada para describir las vidas posibles de los objetos asociando un término del álgebra para representar el conjunto de trazas permitidas en la vida del objeto.

Las **restricciones de integridad** establecen las propiedades que se deben de satisfacer en cualquier estado (en el caso de restricciones estáticas) o a lo largo de una secuencia de estados (cuando hablamos de restricciones dinámicas). Se representan usando lógica de 1er orden y temporal, respectivamente.

Finalmente, La **interfaz** pública de un objeto está compuesta por el conjunto de atributos observables y/o eventos que el objeto ofrece a la sociedad. El proceso de exportar propiedades se soporta en OASIS a través de la noción de vista o interfaz de proyección. La definición de vistas establece un modelo de protección que permite a los diseñadores especificar el aspecto que las clases ofrecen al resto del sistema. OASIS ofrece, además de las vista de proyección tres tipos más de vistas: restricción de poblaciones, a medida y derivadas.

En el apéndice A se puede observar un ejemplo de especificación OASIS sencilla que, además, sirve como punto de partida para la práctica final.

[1] También llamados instancias.

3. Delphi como herramienta de implementación

Las razones que nos impulsaron a escoger Delphi, de entre los diferentes entornos de producción de software, para realizar la implementación de los conceptos OASIS fueron:

1. Delphi permite el desarrollo rápido de aplicaciones[2] y proporciona un entorno visual cómodo y sencillo para la construcción de aplicaciones de uso general que puede ser útil en la vida profesional de los alumnos en el futuro.
2. El lenguaje de programación de Delphi (*Object-Pascal*), es un Pascal extendido para incorporar los conceptos objetuales, lo cual hace que la distancia entre los conceptos a implementar y el lenguaje destino sea la menor posible.
3. El modelo de objetos que se usa en Delphi creemos que es sencillo y didáctico, y permite centrarse en los objetivos de la práctica, sin perderse en las complejidades que implica el uso de otros lenguajes de programación OO.

Cuando los alumnos llegan a está asignatura no conocen nada sobre Delphi, aunque si tienen conocimientos sobre análisis y diseño OO porque han sido presentados en otras asignaturas. Es por ello, que las primeras 5 prácticas se dedican a la presentación del entorno.

1ª Práctica: Introducción a Delphi (2h)

Se presenta el Entorno de Desarrollo Integrado de Delphi, con sus diversas ventanas y áreas de trabajo. Se enseña como construir una primera aplicación en Delphi, todos los tipos de ficheros que usa y genera el entorno: ficheros de proyecto, módulos de implementación, formularios, etc. Se les muestra el depurador integrado que incorpora. Y, finalmente, se hace una presentación de los formularios, modos de apertura y propiedades fundamentales.

2ª Práctica: El Modelo de Objetos de Delphi (2h)

Se presenta, con todo lujo de detalles y ejemplos, el modelo de objetos de Delphi. La presentación incluye:

1. Definición de clases a través de los atributos y métodos que soportan.
2. Detalles de la implementación del modelo de objetos de Delphi.
3. Uso de constructores y destructores.
4. Encapsulación de la información y su relación con los módulos.
5. Atributos y métodos de clase.
6. Punteros a métodos.
7. Herencia y compatibilidad.
8. Sobreescritura (métodos dinámicos *versus* virtuales)
9. Métodos abstractos.
10. Gestión de excepciones
11. *Run-Time Type Information.* Propiedades.

3ª Práctica: Implementación en *Object-Pascal* (4h)

Con el objetivo de afianzar los conocimientos adquiridos en las sesiones anteriores, se pide que se construya una aplicación informática que permita a dos personas jugar al ajedrez. El programa debe mostrar el tablero e ir preguntando alternativamente al jugador que lleva las blancas o las negras su jugada. Si la jugada es incorrecta, se debe repetir la pregunta. Cuando un rey sea 'comido' la partida acaba.

Ya que el objetivo es que practiquen con el modelo de objetos, no se pretende que la representación por pantalla del tablero sea gráfica, sino que los alumnos se centren en la funcionalidad, por lo que se recomienda que implementen una aplicación textual haciendo uso de los comandos: *write*, *writeln*, *read* y *readln*. Un posible ejemplo de interfaz es el siguiente:

```
  7 Tn    An Nn Rn An Cn Tn
  6 Pn Pn Pn Pn    Pn Pn Pn
  5       Cn
  4    Ab       Pn
  3             Pb
  2                Cb
  1 Pb Pb Pb Pb    Pb Pb Pb
  0 Tb Cb Ab Nb Rb       Tb
    0  1  2  3  4  5  6  7

            y x y x
Jugada n: 6 0 5 0
```

[2] *RAD* (*Rapid Application Development*)

4ª Práctica: Bases de Datos en Delphi (2h)

Como la aplicación que deben construir para la evaluación de la asignatura debe soportar objetos persistentes, es necesario que los alumnos conozcan los mecanismos que les ofrece Delphi para salvar la información en bases de datos relacionales.

La 4ª práctica está dedicada a las bases de datos en Delphi y la presentación incluye:

1. Utilidades incluidas en Delphi para la gestión de bases de datos.
2. Componentes disponibles de Bases de Datos, tanto los que acceden a la información *Data-Access*, como los que permiten su visualización en los formularios *Data-Controls*.
3. Ejemplos de accesos a tablas y consultas SQL.
4. Propiedades relevantes de los *Data-Controls*.
5. Uso de parámetros en sentencias SQL dinámicas.
6. Componentes *TField*.
7. Navegación por los registros a través de programa. *Bookmarks*.
8. Edición, inserción y modificación de registros por programa.
9. Uso de rangos.
10. Uso de los *Wizard*.
11. Eventos de *TTable*.
12. *Data Modules*.
13. Transacciones.

5ª Práctica: Construcción de una Aplicación que use BD en Delphi (4h)

Se pretende que los alumnos adquieran destreza en la creación, uso y mantenimiento de aplicaciones que accedan a bases de datos, dentro del entorno de programación de Delphi.

Para ello deben construir una aplicación que se encargue de la venta de artículos de ropa a los clientes. La aplicación debe permitir introducir los datos de los nuevos artículos, mantener el almacén de los artículos disponibles actualmente, el mantenimiento de los clientes, así como la venta de artículos a los clientes generando facturas.

Creemos que en el futuro los alumnos llegarán a esta asignatura con mayores conocimientos sobre Delphi ya que las asignaturas que se imparten en cursos anteriores dentro de la unidad docente están siendo sometidas a un proceso de revisión y van a introducir Delphi en diversas asignaturas, con lo que las primeras prácticas se podrán reducir o incluso eliminar para centrarnos en lo que es el objetivo de las prácticas.

4 Traducción de Esquemas Conceptuales OASIS a Programas Delphi

El objetivo último de las prácticas de TSA es ayudar a los alumnos para que alcancen una cierta capacitación en la implementación de modelos OO complejos en entornos de producción de software modernos, de forma que fueran capaces de aplicar unos procesos de traducción automáticos.

Dentro del grupo de investigación se han realizado algunos trabajos relacionados con el tema de la práctica ([5] y [6]). El tema también ha sido tratado por otros grupos de investigación como se puede ver en [3]. La implementación en diferentes lenguajes de programación partiendo de diseño en OMT aparece reflejado en el libro de Rumbaugh [7].

A modo de resumen, de todos los artículos anteriores y algunos otros, se propone la siguiente práctica.

6ª Práctica: Implementación de Conceptos OASIS en Delphi (8h)

Consiste en la implementación en Delphi del esquema conceptual OASIS de una biblioteca (en el apéndice A se puede ver la especificación). La aplicación debe permitir el uso de objetos OASIS persistentes, consultar el valor de los atributos de los objetos, disparar eventos, crear y destruir objetos, etc. En resumen, la aplicación debe ser funcionalmente equivalente a la especificación OASIS.

Para la traducción a Delphi de la especificación OASIS se les proporcionan las siguientes guías de trabajo:

La arquitectura de la aplicación debe tener tres capas:

1. Lógica de Presentación: formada por las ventanas o formularios que presentan la información al usuario y permiten que exprese sus necesidades a través del teclado o ratón,
2. Lógica de Negocio: entendida como el conjunto de objetos OASIS que

proporcionan la funcionalidad de la aplicación.

3. Lógica de Datos: proporcionada por el SGBD que se utiliza para almacenar la información persistente de los objetos.

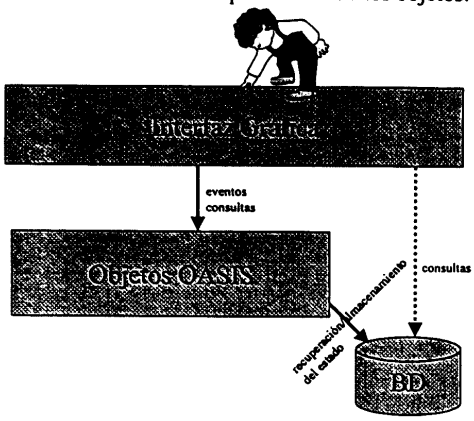

Figura 1: Arquitectura de tres capas propuesta.

Aunque la lógica de la aplicación no debería acceder a los datos almacenados directamente[3], se les permite la consulta de información de las tablas con fin de presentar la información en pantalla de forma sencilla usando los componentes que Delphi proporciona específicamente para ello. Eso sí, se insiste en que es únicamente con fines de observación y que la manipulación del estado de los objetos se debe realizar a través de la invocación de los eventos que proporcionan los objetos OASIS.

Los objetos que implementen deben ser persistentes, de forma que deben salvar el estado en tablas relacionales usando algún esquema de traducción de OASIS a tablas relacionales. El siguiente esquema ilustra un posible esquema para almacenar los objetos relacionando conceptos OASIS y conceptos del modelo relacional.

OASIS	M. RELACIONAL
una clase	una tabla
un objeto	una tupla
atr. ctes y vbles.	columnas de la tabla
atr. derivados	funciones Delphi o campos calculados en los componentes *TTable* o campos almacenados en las tablas
identificador	clave primaria
atr ctes.	modificador `NOT NULL`
restricciones de integridad estáticas.	restricciones de la tabla `CHECK`
agregaciones 1:M	clave ajena
agregaciones N:M	tabla nueva clave primaria composición de las comp.
especialización	tabla con los atributos emergentes y clave primaria la del padre o unir todas las posibles especializaciones en una tabla

Además de tener el estado de los objetos almacenado en tablas, es necesario construir objetos Delphi que se comporten como lo harían los objetos OASIS.

Para ello, por cada clase OASIS habrá una clase Delphi con unos atributos que serán los atributos de los objetos OASIS y unos métodos que implementarán la funcionalidad de los eventos de los objetos OASIS.

Cuando se invoque algún servicio de algún objeto, lo primero que se deberá hacer será localizar el registro de la tabla donde se encuentre almacenado.

Los atributos constantes y variables pueden obtener su valor de los valores almacenados en los campos de la tabla. Los atributos derivados se pueden implementar como campos calculados en los componentes *TTable* de forma que se comporten exactamente igual que los otros atributos.

Los eventos deben implementar el modelo de ejecución siguiente:

1. Comprobar que el evento es válido según las vidas posibles que se hayan definido en el párrafo *process* de la especificación.
2. Comprobar la validez de la precondición del evento.

[3] Desde un punto de vista estricto, el estado de los objetos debería ser consultarlo a través de los atributos que les proporcionaran los objetos OASIS.

3. Modificar el estado del objeto basándose en las *valuations*.
4. Comprobar que se siguen manteniendo las restricciones de integridad definidas.
5. Disparar los *triggers* que se hayan podido activar como consecuencia del cambio de estado[4].

Para que los métodos de las clases Delphi puedan saber sobre que objeto OASIS deben ejecutarse se debe pasar como argumento el identificador del objeto afectado.

Se puede implementar la funcionalidad del párrafo *process* añadiendo un nuevo campo a las tablas que permita guardar en qué estado se encuentra el objeto, y una tabla de transiciones con los estados como filas y los eventos como columnas. La tabla contendrá en la intersección de las filas y columnas el estado que se alcanza si encontrándose el objeto en el estado S llega el evento E, error si no se acepta el evento o fin si el evento es el de destrucción del objeto. Realmente, se implementa un autómata finito determinista, siendo la tabla una representación de la función de transición entre estados.

Para implementar los eventos compartidos se debería montar una arquitectura con múltiples hilos de ejecución en el que cada objeto tuviera su hilo, sincronizar los procesos, etc. Para simplificar la práctica, se ha optado por serializar la ejecución de los eventos en los diferentes objetos y hacer uso de las transacciones para simular el comportamiento síncrono. Cuando se tenga que ejecutar un evento compartido que implique a más de un objeto se abre una transacción, confirmándola si todo va bien o abortándola si ocurre algún problema. En la práctica el evento prestar es compartido entre las clases libro y socio.

Uno de los mayores problemas en la implementación se encuentra en el concepto de especialización temporal. En los lenguajes de programación clásicos no se puede cambiar el comportamiento ni el estado de los objetos en tiempo de ejecución. Para simularlo, se puede hacer uso de un objeto interfaz que reciba todos los mensajes que vayan dirigidos a cualquier objeto de una jerarquía de herencia y los redirija a objetos diferentes en función de la especialización que en ese momento tenga el objeto. En la práctica, los socios tendrán un socio interfaz que recibirá todos los mensajes dirigidos al socio y lo redirigirá al socio_fiable si no está especializado y al socio_no_fiable si el objeto socio se encuentra especializado en ese momento.

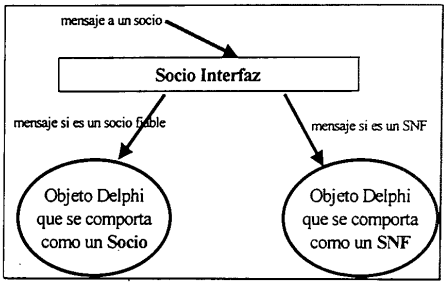

Figura 2: Implementación para la simulación de la especialización temporal.

Como resultado del trabajo realizado, los alumnos deben presentar:

- Una memoria donde se recoja el diseño de la aplicación, el manual de usuario y los mecanismos utilizados para trasformar conceptos OASIS en DELPHI.
- Un disco con la memoria, el código fuente y el ejecutable de la aplicación.

5 Conclusiones

Se ha presentado un ejemplo de aprovechamiento de los resultados del trabajo de investigación en la docencia que se imparte en la *Facultad de Informática de Valencia*.

El resultado de la experiencia ha sido muy satisfactorio: del total de alumnos (52) que había matriculados, el 73% finalizó la práctica. De los que entregaron la práctica, el 82% fueron calificados como aptos con un total de 7 sobresalientes, 19 notables y 5 aprobados.

Creemos que es interesante trasmitir a los alumnos los conocimientos que ha recopilado el grupo de investigación fruto de sus trabajos, ya que,

[4] Implementaciones más complejas, hacen uso de monitores de *triggers*, pero para la práctica actual es suficiente con un mecanismo más sencillo.

normalmente, son conceptos avanzados a los cuales los alumnos difícilmente tendrían otro acceso.

Además, el grupo de investigación se da a conocer a los alumnos y éstos encuentran un posible marco en el que realizar, con posterioridad, su PFC. En la actualidad, varias personas están realizando su PFC en temas relacionados con la práctica como parte de una herramienta CASE que genere código automáticamente a partir de las especificaciones OASIS haciendo uso de diferentes esquemas de traducción. Otras tres personas se encuentran realizando su PFC dentro del grupo en temas muy relacionados con OASIS y sus extensiones.

Bibliografía

[1] Carsí J.A., Ramos I., Penadés M.C., Pelechano V., *Descripción del modelo de objetos de OASIS a través de la Transaction Frame Logic*, I Jornadas de Trabajo en Ingeniería del Software (JIS'96), Sevilla, pp.85-96, Noviembre 1996.

[2] Canós J.H., *OASIS: Un lenguaje único para Bases de Datos Orientadas a Objetos*, Tesis Doctoral, Octubre 1996, DSIC-UPV (Valencia).

[3] Durán A., Toro M., Amaya A., *Design of an Automatic Generator of Object-Relational Persistency Mechanisms*, II Jornadas de Ingeniería del Software (JIS'97), San Sebastian, 1997.

[4] Pastor, O., Ramos, I., *OASIS version 2 (2.2): A Class-Definition Language to Model Information Systems using an Object-Oriented Approach*, SPUPV-95.788, Universitat Politècnica de València, 1996.

[5] Pastor O., Insfrán E., Pelechano V., *Modelado OO Aplicado a Entornos de Desarrollo Relacionales*, II Jornadas de Investigación y Docencia en Bases de Datos (JIDBD'97), Madrid, Julio 1997.

[6] Romero J., Ramos I., Pastor O., *Delphi's Object Model*, Report Técnico II-DSIC-37/97, Octubre 1997.

[7] Rumbaugh J., et al., *Modelado y Diseño Orientado a Objetos*, Prentice Hall, ISBN 013-240698-5, 1996, Capítulos 4, 15, 17.

Apéndice A: Especificación OASIS del Sistema de Información de una biblioteca

En la biblioteca existen libros que pueden ser prestados a unos socios. Existen unos bibliotecarios que son los encargados de efectuar los prestamos, así como, dar de alta tanto nuevos libros como socios. Cada socio puede tener prestados como máximo 10 libros simultáneamente. Si los socios incurren en alguna falta se les marca, de manera que, como socios marcados, sólo pueden sacar un máximo de dos libros. Los socios no fiables son perdonados, automáticamente, en el caso de que devuelvan todos los libros que tienen en préstamo.

```
conceptual_schema biblioteca

domains Nat;Bool;String;Date.

class Libro
 identification
  by_codigo_libro:(codigo_libro);
 constant_attributes
 codigo_libro : String ;
 titulo : String ;
 tema : String ;
 variable_attributes
  disponible : Bool ;
 private_events
  alta_libro () new;
  baja_libro () destroy;
 shared_events
  devolver () with Socio;
   prestar () with Socio;
 valuations
  [alta_libro ()] disponible=TRUE;
  disponible=TRUE [prestar ()]
                  disponible=FALSE;
  disponible=FALSE [devolver ()]
                   disponible=TRUE;
 preconditions
  baja_libro() if disponible=TRUE;
  prestar () if disponible=TRUE;
  devolver () if disponible=FALSE;
 process BIB:Bibliotecario;
  Libro = BIB:alta_libro() Libro1;
  Libro1 = BIB:baja_libro() +
           BIB:prestar () Libro2;
  Libro2 = BIB:devolver () Libro1;
end_class

class Socio
 identification
  by_codigo_socio :(codigo_socio);
 constant_attributes
  codigo_socio : Nat ;
  nombre : String ;
 variable_attributes
  num_libros : Nat ;
```

```
   avisado : Bool ;
  private_events
   alta_socio () new;
   baja_socio () destroy;
   fichar ();
  shared_events
   prestar () with Libro;
   devolver () with Libro;
  constraints
   static
    num_libros <= 10;
  valuation
   [alta_socio ()] avisado=FALSE &
                   num_libros=0;
   avisado=FALSE [fichar ()]
                    avisado=TRUE ;
   [prestar ()]
           num_libros=num_libros+1;
   [devolver ()]
           num_libros=num_libros-1;
  preconditions
   baja_socio () if num_libros=0.
  process BIB:Bibliotecario;
   Socio = BIB:alta_socio() Socio1;
   Socio1 = BIB:baja_socio() +
            ( BIB:prestar() +
              BIB:fichar()  +
              BIB:devolver() )
            Socio1;
end_class

class Bibliotecario
 identification
 by_codigo_biblio:(codigo_biblio);
 constant_attributes
  codigo_biblio : Nat ;
  nombre : String ;
 private_events
  alta_biblio () new;
  baja_biblio () destroy;
 process BIB:Bibliotecario
  Bibliotecario=BIB:alta_biblio()
                Biblio1;
  Biblio1 = BIB:baja_biblio ();
end_class

complex class Prestamo aggregation of
Libro(relational, static, univalued,
disjoint, flexible,not null),
Socio(relational, static, univalued,
nodisjoint, flexible, not null)
 identification
  by_codigo_prestamo:(codigo_prestamo
  );
 constant_attributes
  codigo_prestamo : Nat ;
  fecha : Date ;
 derived_attributes
  cant_dias : Nat ;
 private_events
  prestar () new;
   devolver () destroy;
 derivation
  cant_dias = Date() - fecha
 process BIB:Bibliotecario;
  Prestamo = BIB:prestar()Presta1;
  Presta1 = BIB:devolver ();
end_class

complex_class SNoFiable
specialization_of Socio.
 constant_attributes
  fecha : Date ;
 private_events
  fichar() new;
  perdonar destroy;
 valuation
  avisado=TRUE [perdonar()]
                avisado=FALSE ;
 preconditions
  prestar () if num_libros<3.
 processes BIB:Bibliotecario,
           SNF:SNoFiable;
  SNoFiable = BIB:fichar() SNF1;
  SNF1=( BIB:prestar()+
         BIB:devolver() ) SNF1 +
       BIB:perdonar() +
       SNF:perdonar();
 trigger
  self::perdonar() if num_libros=0
end_complex_class

end_conceptual_schema
```