UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

Colegio de Ciencias y Humanidades.

Unidad Académica de los Ciclos

Profesionales y de Posgradado

Instituto de Investigaciones en Matemáticas

Aplicadas y Sistemas

ESPECIFICACION FORMAL DE UN MODELO CONCEPTUAL PARA BASES DE DATOS ORIENTADAS A OBJETOS

T E S I S

Que para obtener el grado de

MAESTRO EN CIENCIAS DE LA COMPUTACION

P r e s e n t a

EMMA CECILIA MONTERO MEJIA

Mayo 1987

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

INDICE

BIBLIOGRAFIA.

INTRODUCCION

El objetivo de esta Tesis es la definición de una especificación formal de un modelo conceptual para bases de datos orientadas a objetos.

Consideramos que el tema de este trabajo es de importancia principalmente para los investigadores en las áreas de bases de datos y de especificación formal por las razones que se exponen adelante. Sin embargo, para la comprensión del trabajo no se requiere ser experto en dichas áreas, basta tener conocimientos básicos de bases de datos y matemática elemental (lógica y conjuntos).

Actualmente son reconocidas las limitaciones de los modelos de bases de datos tradicionales en los problemas que plantean los sistemas de procesamiento de información modernos.

Esto ha motivado el surgimiento de una nueva corriente en las investigaciones en Bases de Datos que pretende resolver los requerimientos actuales. Algunos investigadores del área interesados en el tema han trabajado en aspectos relacionados desde hace relativamente poco tiempo.

Los resultados de sus investigaciones se encuentran dispersos y en algunos puntos existe todavía una definición poco clara, además, puede decirse que ha habido muy poco trabajo en aplicaciones prácticas.

En el IIMAS el Dr. Alejandro Buchmann ha impulsado

el desarrollo e implementación de un modelo de bases de datos aplicado a un sistema CAD,que responda a las necesidades actuales.

En este contexto, se pretende aprovechar la experiencia proporcionada por el Dr. Alejandro Buchmann y la Dra. Concepción Perez de Celis para el desarrollo de una definicion formal de un modelo al cual hemos llamado "modelo orientado a objetos" motivado por su trabajo. Esperamos que los resultados de esta tesis aporten elementos que retroalimenten a los interesados.

Así, el modelo que se estudia en esta Tesis ha ido desarrollándose con el objeto de superar algunas limitaciones semánticas de los modelos anteriores, como son: prohibición de anidamientos, falta de manejo de herencia, imposibilidad de manejo de objetos complejos y sus componentes, etc.

La especificación formal pretende presentar de manera muy sistemática, los fundamentos del Modelo; establecer en forma clara, sin inconsistencias ni ambigüedades los conceptos principales del mismo y coadyuvar así en el establecimiento de líneas de investigación, desarrollo e implementación en este tipo de bases de datos.

Es importante destacar que, mientras el desarrollo de formalizaciones respecto de los sistemas tradicionales se ha realizado --aunque no en forma muy abundante-- en esta nueva área, el problema de formalización no ha sido trabajado aún. Estamos planteando así en esta Tesis, la solución a un problema abierto en la actualidad.

En el capítulo I de este trabajo se discuten los preliminares necesarios en cuanto a conceptos que se utilizarán y antecedentes para bases de datos orientadas a objetos, que ayudan a identificar su importancia e ilustran el tipo de aplicaciones que motivan su desarrollo.

La descripción informal del modelo orientado a objetos se expone en el capítulo II. Conviene apuntar que esta descripción se elaboró en base a la especificación formal, aunque en este capítulo no se introduce aún ninguna formalización ya que se prefirió presentar primero una visión global e intuitiva del modelo.

Las dos primeras secciones del Capítulo III son introductorias al área de especificación formal y están dirigidas sobre todo a los lectores que no conocen el tema. La tercera sección de este capítulo introduce el lenguaje de especificación que se ha elegido para la formalización y que tiene el nombre de "Z". Es necesario revisar con cuidado esta parte para comprender los capítulos siguientes.

Los capítulos IV, V y VI presentan la especificación formal desarrollada y constituyen la parte medular de este trabajo.

CAPITULO I.

ANTECEDENTES PARA BASES DE DATOS ORIENTADAS A OBJETOS.

Los conceptos y terminología básicos que usaremos serán presentados en la primera sección de este capítulo.

En el desarrollo teórico y práctico de bases de datos han tenido una importancia fundamental tres modelos básicos a lo cuales hemos llamado "modelos tradicionales". En la sección I.2 se presentan muy brevemente estos modelos para después exponer cómo dichos modelos no satisfacen las necesidades que plantean algunas aplicaciones modernas. En la sección siguiente se presenta una de esas aplicaciones y se explica en qué puntos fallan los modelos tradicionales para poder aplicarse en esta área, de donde se justifica la necesidad de otros modelos.

La siguiente sección (I.4) presenta las propuestas más conocidas a las limitantes de los modelos tradicionales ; algunas de las características de estos modelos se incorporarán al modelo orientado a objetos, como se verá en el capítulo II.

I.1 DISCUSION GENERAL. QUE ES UNA BASE DE DATOS

No creemos que sea necesario repetir aquí extensivamente lo que los textos básicos plantean en cuanto a los

conceptos fundamentales del tema dado que esta información se encuentra accesible en nuestro medio y eso nos dispersaría del objeto del presente trabajo. Sólo puntualizaremos el significado asumido para cada concepto que se utiliza con el objeto de evitar ambigüedades; es común en esta área el manejo de un mismo vocabulario para conceptos diferentes.

Una base de datos es una colección de datos almacenados de acuerdo a una cierta estructura particular. Este conjunto de datos es utilizado en sistemas de aplicación para propósitos determinados (estadísticos, censales, etc. etc).

En la literatura se encuentra generalmente una división de la arquitectura de un sistema de bases de datos -- para facilitar su estudio -- en tres niveles (de acuerdo a los estandares establecidos por ANSI/SPARC (1)):

- Nivel interno. Se refiere a la forma en que se da el almacenamiento físico de los datos.

- Nivel externo. Se refiere a la forma en que los datos son vistos por los usuarios individuales.

- Nivel conceptual. Se refiere a la representación abstracta de la estructura de la base de datos en su conjunto.

A cada uno de los niveles anteriores, ANSI/SPARC les

asocia su respectivo modelo (se habla así de los modelos interno, externo y conceptual).

Puesto que el desarrollo medular de este trabajo sólo tiene relación con el nivel conceptual, nos centraremos en la discusión de lo que se refiere al modelo conceptual.

El modelo conceptual puede definirse a grandes razgos como una representación abstracta del contenido de información completo de una base de datos.

En [DATE 77] se da el nombre de "modelo de datos" al modelo conceptual (2). Es importante aclarar que el concepto de modelo de datos se ha ido desarrollando paulatinamente a partir de 1970, cuando fué formulado por vez primera por Codd (3).

En la actualidad lo que se entiende por modelo de datos es lo siguiente (de acuerdo al concepto que se maneja en [TSICH 82]).

Un modelo de datos consiste de dos partes: un conjunto de reglas generales para la especificación de las estructuras de datos y un conjunto de operaciones permitidas sobre los datos.

El principal objetivo de un modelo de datos es proporcionar un sistema formal para la representación y manipulación de la información, que sirva de base para la arquitectura de todo el sistema de base de datos. El modelo de datos es la parte medular de un sistema de base de datos y es el principal elemento para caracterizar sus propiedades.

En este trabajo, la "especificación formal del modelo conceptual" a que se refiere el título debe tomarse como

la especificación formal del modelo de datos, entendiendo por modelo de datos lo que se define arriba. En adelante llamaremos algunas veces simplemente 'modelo' al modelo conceptual o modelo de datos.

Un esquema de base de datos (EBD) define las estructuras y propiedades admitidas para los datos de un determinado modelo y también especifica las asociaciones permitidas entre los elementos de la base de datos (1). A un EBD se encuentra asociado el llamado lenguaje de definición de datos (data definition language (DDL)) que es una expresión concreta del EBD por medio del cual se proporciona la sintaxis para la declaración de las variables y asociaciones de la base.

Una instancia de una base de datos (IBD) es un determinado conjunto de valores que corresponde a los elementos y satisface las restricciones de asociación definidos en el EBD.

Ahora ya manejamos los términos necesarios para volver sobre el concepto de base de datos y podemos redefinirlo más especificamente:

Una base de datos (BD) es una IBD que corresponde a un EBD determinado.

Para tener un modelo de datos completamente definido falta sólo referirnos con más presición al conjunto de operaciones permitidas sobre los datos. Estas operaciones en realidad lo que establecen son las acciones admitidas en una BD determinada para poder llegar a otra BD con

la característica de que su EBD es el mismo pero su IBD podría ser diferente al inicial. Así, estamos asumiendo que el EBD se mantiene inalterado para toda operación definida en el modelo.

Al conjunto de operaciones se encuentra asociado, a nivel de implementación , el llamado lenguaje de manipulación de datos (data manipulation language (DML)) el cual es una expresión concreta de las operaciones por medio del que se proporciona una sintaxis para trabajar con los operadores del modelo.

Por último, un sistema manejador de base de datos (SMBD) (SMBD) es una implementación de un DDL y un DML específicos, además de que puede o debe incluir facilidades para la definición del esquema específico.

Aunque el llegar a establecer una implementación de un sistema manejador de BD escapa a los objetivos de esta tesis, sí se pretende que a partir del trabajo que se desarrolla en los siguientes capítulos se puedan establecer bases firmes para una buena implementación del modelo aquí propuesto.

I.2 MODELOS DE DATOS TRADICIONALES.

Los siguientes modelos que se describen de manera general e informal son los considerados como básicos. Se exponen aquí pues es común comparar a los modelos más recientes con dichos modelos que cronológicamente fueron los

primeros que se desarrollaron.

I.2.1 MODELO RELACIONAL.

El modelo relacional [5] consiste de

1) Una colección de relaciones tabulares que varían en el tiempo (las cuales cumplen ciertas propiedades sobre llaves y dominios).

2) Reglas de integridad para las operaciones de inserción, actualización y borrado.

3) Algebra relacional.

El punto 1) se refiere a las estructuras de datos y a las asociaciones posibles entre ellos. Las entidades se definen mediante relaciones. Las asociaciones ente entidades también se representan en este modelo por relaciones, lo cual implica que entre las relaciones no hay ligas sino que, las asociaciones entre ellas se representan por valores en tablas.

La definición de relación en este modelo es casi la misma que la definición matemática del concepto, la única diferencia es que las relaciones tabulares del modelo pueden variar en el tiempo.

El punto 2) se refiere a las operaciones sobre los datos (las operaciones de inserción, actualización y borrado son básicas en cualquier modelo) y el punto 3) puede

ser asumido debido a los fundamentos matemáticos que van implícitos en el concepto de relación.

Debemos subrayar que el concepto básico de este modelo es el de relación (aunque esto pueda parecer muy obvio). Dado que las relaciones tabulares consisten de un conjunto de eneadas que pertenecen a la relación y puesto que el concepto matemático de conjunto no admite la contención de dos elementos idénticos, podemos entonces observar que una característica del modelo relacional es que no admite la existencia de dos eneadas de una relación que sean idénticas (que tengan los mismos valores), es decir, no puede haber instancias de datos duplicadas.

La característica anterior implica que toda eneda tiene al menos una llave (en este modelo una llave es cualquier subconjunto de valores de una eneada que la identifique de manera única).

Otra característica de un conjunto (y por lo tanto de una relación tabular) es que el orden de sus elementos no es significante. Por lo tanto, en el modelo relacional el orden de las enenadas no es significante.

El modelo relacional nos proporciona bastante libertad para modelar los datos, sin embargo, un grave problema es que la forma en que deben ser manejadas las asociaciones entre relaciones ocasiona que se pierda el contenido semántico de ellas (6), además, este modelo es inadecuado para la representación de asociaciones de tipo jerárquico. Otro problema es que este modelo obliga en varios casos a

incluir relaciones adicionales (por ejemplo para la representación de relaciones de muchos a muchos) las cuales dificultan la interpretación del significado de las aplicaciones y muchas veces parecen muy artificiales.

I.2.2 MODELO DE REDES.

En el modelo de redes (7) los datos son representados por registros. Las asociaciones entre datos se representan mediante ligas entre registros, las cuales deben ser funcionales.

La estructura del modelo es similar a una gráfica donde los nodos corresponden a los registros y los arcos corresponden a las ligas. Los arcos son dirigidos, su dirección es opuesta a la dirección de funcionalidad. El registro al que se dirige el arco es el miembro, el registro propietario es del cual surge dicho arco. Las ligas, por lo general, tienen un nombre o tipo.

En este modelo es posible representar correspondencias de muchos a muchos entre registros. Para que esto sea posible sin violar la funcionalidad de las ligas hay que usar registros auxiliares como conectores.

Las operaciones sobre los datos son un poco más complicadas que en el modelo relacional. Una red más elaborada requiere de más operadores para manejarla.

Son características que nos interesan en este modelo

las siguientes:

- Un registro miembro puede tener a lo más un registro propietario.
- Un registro propietario puede tener cualquier número de miembros.
- Un mismo registro no puede ser miembro en más que una liga (del mismo tipo).
- No se permiten ligas recursivas (una liga recursiva es la que tiene como propietario y miembro al mismo registro).

Algunos plantean que el problema principal del modelo de redes es su complejidad. Sin embargo, hay otra desventaja que nos parece más importante y es que en este modelo puede haber cierta ambigüedad en la interpretación del significado de las ligas (8).

I.2.3 MODELO JERARQUICO.

En la mayoría de los textos se trata al modelo jerárquico como un caso particular o restringido del modelo de redes.

En este modelo, al igual que en el de redes, los datos son representados por registros. Las asociaciones entre los datos también se representan por ligas funcionales entre los registros.

Una particularidad del modelo jerárquico es que su estructura tiene la configuración de un árbol ordenado,

donde los nodos corresponden a registros y las ramas a ligas. La dirección de las ramas o ligas es opuesta a la dirección de funcionalidad. El registro al cual se dirige una rama es llamado hijo, el padre será entonces el registro del cual surge dicha rama.

En este modelo no es posible representar directamente asociaciones o correspondencias de muchos a muchos entre registros (aunque existen varios métodos que permiten superar algunas limitaciones impuestas por esta restricción).

Las operaciones básicas que se efectúan sobre un modelo jerárquico deben satisfacer más reglas de integridad que en los modelos anteriores.

Las características mencionadas en el modelo de redes son también características de este modelo. La principal desventaja del modelo es que, dado que sólo permite una sola liga entre registros, no es necesario etiquetar a dicha liga, con lo cual se pierde el significado de lo que ésta representa en el modelo. Además, la rigidez del modelo ocasiona que a veces sea necesario un cambio en el modelo conceptual de una particular base de datos lo cual puede llevar a confusiones y errores.

Como síntesis de las tres últimas secciones podemos hacer algunas observaciones.

La principal diferencia entre los modelos vistos es la forma en la que pueden representarse y manipularse las asociaciones entre los datos.

Hay una característica que presentan los tres modelos y que en este trabajo nos interesa subrayar. Dicha característica común es la deficiencia que los modelos presentan en la preservación explícita de la semántica correspondiente a la base de datos modelada.

Esta característica es de particualr importancia en algunas aplicaciones de bases de datos como veremos en la siguiente sección. Por lo anterior, desde hace algunos años se vienen desarrollando trabajos encaminados a la solución de este tipo de problemas. Algunas propuestas hechas se presentan en la sección I.4.

I.3 SOBRE LA APLICACION DE BASES DE DATOS A CAD.

Desde la década de los 70's se ha venido difundiendo y desarrollando la asistencia de la computadora en el diseño ingenieril industrial de todo tipo de objetos. Los sistemas realizados en esta área son conocidos con el nombre de CAD (corresponde a la iniciales de Computer Aided Design).

Los sistemas CAD vinieron así a facilitar una gran variedad de tareas en el diseño de infinidad de productos industriales como automóviles, aeroplanos, maquinarias y sus partes, circuitos integrados, además de todo tipo de diseños arquitectónicos, eléctricos, mecánicos y hasta diseño de plantas de procesamiento químico. Todo lo anterior nos da una idea de la importancia de CAD en la sociedad moderna.

En una etapa inicial la tendencia fue el desarrollo de paquetes para diversas tareas específicas. Sin embargo, al requerirse la aplicación de una serie determinada de paquetes sobre los mismos datos, se generaban cantidad de problemas de compatibilidad y comunicación.

Es precisamente el resolver los problemas que surgen del manejo de información compartida lo que constituye una función primordial de los sistemas de BD. Por lo tanto, la necesidad de un manejo de datos más eficiente ha suscitado un creciente interés por la aplicación de bases de datos a CAD.

Es un hecho que la comunidad que trabaja en CAD ha reconocido que las bases de datos son la solución ideal para un adecuado funcionamiento cuando varios paquetes deben ser aplicados sobre un conjunto de datos en común (a su vez, la comunidad de investigadores en bases de datos reconoce en CAD una de las aplicaciones más interesantes de su área). La aplicación de bases de datos en CAD ofrece ventajas como las siguientes.

- Incremento en la consistencia de los datos.
- Mayor integridad de los datos por un decremento en la redundancia de los mismos.
- Protección de la información compartida a través del control de integridad, autorización, concurrencia y recuperación.

Otras ventajas adicionales para el usuario son.

- Respuesta eficiente a sus requerimientos a través

del procesamiento global y compartido de la información.

- Descarga sobre el sistema de las tareas tediosas relacionadas con el almacenamiento físico de los datos (antes a cargo del usuario).
- Aumento en la rapidez de comunicación entre los diferentes paquetes utilizados.
- Acceso integrado a una gran variedad de información heterogenea.
- Proporciona facilidades para lograr nuevas aplicaciones o requisitos especiales, lo que aumenta la flexibilidad en el manejo de la información.

No obstante que la utilización de bases de datos en CAD resuelve problemas muy importantes y proporciona las ventajas señaladas, resulta que para que esto sea efectivamente una realidad, las bases de datos deben cumplir una serie de características.

En la siguiente sección veremos que tipo de problemas específicos se requieren resolver en un ambiente CAD (características de los datos) y los requisitos correspondientes que debe cumplir una base de datos en dicho ambiente [9]. La exposición de estos puntos nos ilustrará el tipo de razones que motivaron el desarrollo de otros modelos más evolucionados que los tratados en I.2

I.3.1 NECESIDAD DE OTROS MODELOS.

A continuación señalamos varias de las características que presentan los datos en CAD para después puntualizar algunos de los requisitos que deben satisfacer las bases de datos aplicadas en dicha área.

Las siguientes son algunas de las propiedades de los datos en un ambiente CAD.

A) Las estructuras, asociaciones y reglas de control de los datos se determinan previamente y en la gran mayoría de los casos son completamente estáticas. Las instancias o valores específicos de los datos son de naturaleza dinámica.

B) En sistemas CAD muchos de los datos se refieren a los objetos que van a ser diseñados, por lo cual dichos objetos tendrán -vinculados a ellos- restricciones, es decir, deberán satisfacerse ciertas condiciones sobre algunas de las características o atributos del objeto. Además, las asociaciones posibles de un objeto con otros podrán restringirse también ocasionalmente. Ejemplo: Supongamos que el objeto a modelar es una llanta, uno de cuyos atributos es la cantidad de aire que puede contener, este atributo, obviamente, debe estar restringido a la capacidad máxima de aire que la llanta puede soportar.

Este punto es muy importante porque de la capacidad del sistema para modelar caracteríticas semánticas de este tipo, depende en gran medida

su eficacia.

C) Los objetos a manejar en el sistema frecuentemente tienen algunas características especiales como las siguientes:

a) Algunos objetos se podrían definir informalmente como "complejos" debido a que poseen una estructura elaborada, con elementos de una gran variedad.
Ejemplo: Si el objeto a diseñar es una planta industrial tenemos un objeto complejo, con diferentes tipos de objetos constituyentes, con funciones también diferentes.

b) A medida que los objetos son más complejos es más necesario que se registre de alguna forma el significado (la semántica) de cada tipo de asociación que se mantiene entre cada uno de sus constituyentes.
Siguiendo el ejemplo del punto anterior se requiere registrar el tipo de asociaciones entre la planta y sus secciones, entre las secciones y sus funciones, entre las secciones y su equipo, entre el equipo y sus partes, etc. etc.

c) Un objeto complejo puede tener constituyentes que formen parte a su vez (son constituyentes) de otros objetos complejos, es decir, los objetos pueden ser no disjuntos.
Por ejemplo, en la planta industrial puede haber

secciones (objetos complejos) que tengan algún equipo en común.

d) Un objeto complejo puede contenerse como elemento a sí mismo, es decir, puede presentarse la recursividad. Por ejemplo la estructura de una tubería (objeto complejo) podría tener como constituyentes otras tuberías (como brazos o ramas de la tubería central).

D) Naturaleza muy heterogenea de los datos. Las aplicaciones de CAD requieren el acceso a una gran variedad de tipos de información. En las actividades de diseño intervienen objetos de las más diversas características: documentos, dibujos, mapas, gráficas, etc.
Esta información heterogenea debe ser manejada y accesada en forma integrada.

E) Existencia de diferentes categorías de datos determinadas por el comportamiento de sus valores respecto al tiempo. De acuerdo a este enfoque podrían definirse tres categorías:

a) Datos estables. Los datos cuyas instancias son completamente estables, por ejemplo los componentes de un catálogo o las normas de seguridad que debe cumplir un proceso determinado.

b) Datos altamente estables. Los datos cuyas instancias son "casi" estables, por ejemplo el

diseño ya aprobado de algún circuito que sólo podría cambiar en una situación excepcional con la aprobación de todos los ingenieros afectados y autorizados.

c) Datos temporales. Son los datos cuyas instancias son inestables o dependientes del tiempo. Por ejemplo la proposición de alternativas preliminares para un determinado diseño.

F) Las operaciones básicas en un ambiente CAD son las mismas que en toda base de datos: inserción, borrado, actualización y recuperación. Lo que varía es la ocurrencia de estas operaciones que depende de la categoría de los datos.
En datos estables la operación más frecuente es la recuperación. En datos altamente estables se pueden presentar las cuatro operaciones pero las más frecuentes serán la recuperación e inserción.
En el caso de los datos temporales también se pueden presentar las cuatro operaciones pero la menos importante es la recuperación.

Por supuesto que las características que se acaban de listar no son todas las que presentan los datos en un ambiente CAD, sin embargo sí se expusieron algunas de las características más importantes a considerar en la especificación de un modelo para bases de datos que pretenda ser aplicable a CAD (10).

En cuanto a los requisitos que debe satisfacer una base de datos en el ambiente que estamos estudiando, éstos se deducen directamente de las características de los datos arriba enlistadas y se resumen en los siguientes puntos:

A) Capacidad para modelar y manejar conocimiento semántico especializado para el procesamiento de la información, en la forma más explícita posible, ya sea a través de la definición de reglas, condiciones, restricciones semánticas o cualquier otro mecanismo facilitando chequeo de consistencia.

B) Habilidad para representar y procesar objetos complejos, tan estructurados como sea necesario, reflejando las interelaciones de sus constituyentes y de objetos entre sí, de tal forma que se expresen todos los elementos relevantes para las aplicaciones.

C) Simplicidad en la formulación de asociaciones claras semánticamente, donde se permitan unicamente asociaciones explícitas entre los objetos, lo cual evite ambigüedades y facilite el control del sistema.

D) Definición de objetos flexible, permitiendo la recursividad y la representación de objetos no disjuntos entre sí.

Además debe facilitar el manejo de objetos com-

plejos a diferentes niveles de abstracción (el objeto completo, el objeto con algunos de sus - constituyentes, etc.)

E) Modelo de datos sumamente flexible que permita el modelado conceptual de información muy heterogegenea (y posiblemente muy rica en contenido semántico) que comprenda todo tipo de requerimientos ingenieriles.

F) Facilidades para el manejo y representación homogenea e integral de datos de diferentes categorías (con respecto del tiempo en incluso del espacio, como sería el caso de los mapas).

Podemos observar que ninguno de los modelos de datos descritos en la sección I.2 se adapta a las principales necesidades que presentan las aplicaciones en ambiente CAD (11).

En general, se puede afirmar que los modelos tradicionales (también llamados convencionales) no satisfacen los requerimientos que demandan los nuevos sistemas de información, por lo que hay un creciente interés dentro de la comunidad de investigadores del área de bases de datos en el desarrollo de una nueva generación de modelos de datos que respondan a las necesidades modernas. En la siguiente sección veremos algunos de estos modelos.

1.4 MODELOS DE DATOS NO CONVENCIONALES.

Desde hace aproximadamente una década se han presentado propuestas de nuevos modelos de datos que puedan ser útiles en aplicaciones no convencionales (como CAD) y que por lo tanto sean modelos con capacidad expresar y capturar todo el significado relevante de los datos necesario en la aplicación. En general puede decirse que los nuevos modelos buscan la mayor flexibilidad para poder representar el conocimiento del mundo real de la mejor manera posible. Sobre todo, nos interesa presentar en esta sección algunos de los modelos que puedan manejar objetos complejos [12].

A los modelos de datos que a continuación se describen se les conoce en general como modelos semánticos [13] puesto que una de sus características principales es tratar de expresar más riqueza semántica que los modelos convencionales.

En la descripción de cada modelo no se darán detalles ya que el objeto de presentarlos es sólo mostrar algunas de las principales ideas propuestas, sobre todo en lo que concierne al manejo de objetos complejos y a la mayor expresividad semántica. El interesado en profundizar en el estudio de alguno de los modelo podrá remitirse a la bibliografía que se proporciona en cada subsección.

Antes de exponer los modelos semánticos que se consideran de mayor relevancia y que más han influído en investigaciones posteriores vamos a introducir dos conceptos (agregación y generalización) que están presentes de una u otra forma en los tres modelos no convencionales que se desarrollan en las siguientes secciones (14).

I.4.1 AGREGACION Y GENERALIZACION.

Las siguientes ideas, básicas en el desarrollo de los modelos semánticos, fueron definidas en forma explícita en [SMIT 77].

El concepto de abstracción en el contexto de una base de datos, se refiere a las vistas que de los objetos de la base pueden requerirse. Así puede hablarse de diferentes niveles de abstracción de los datos. Por ejemplo, un objeto complejo puede ser visto al nivel de detalle que requiera una aplicación dada, en el caso de la planta industrial (del ejemplo visto en I.3.1) un nivel de abtracción sería ver al objeto (planta industrial) formado sólo por sus componentes más generales (secciones), otro nivel subsiguiente podría contemplar además el equipo de cada sección, y así sucesivos niveles cada vez más detallados.

Las dos diferentes clases o formas de abstracción - en términos de objetos de una base de datos - son la agregación y la generalización.

Agregación es una forma de abstracción de datos por

medio de la cual pueden construirse objetos complejos. Los objetos constituyentes - también llamados objetos agregados - pueden ser, por supuesto, también objetos complejos (es decir, pueden tener sus propios objetos agregados), este proceso sucesivo da lugar a los diferentes niveles de agregación (que posiblemente podrían ser observables a través de diferentes vistas del objeto). La agregación implica la posibilidad de considerar un objeto complejo con tanto detalle acerca de sus constituyentes como sea necesario.

Generalización es otra forma de abstracción la cual permite ver a un dato como un objeto más general. Hay dos tipos de generalización:

- Generalización de instancia a clase (de instancia a tipo).
  Este tipo de generalización (a veces llamado más formalmente como clasificación) generaliza un valor o instancia de un dato a su clase, es decir, se da por membrecía. Por ejemplo, la clase de objeto PERSONA sería una generalización (clasificación) de la instancia 'Luis Pérez'.

- Generalización de clase a clase (de tipo a tipo).
  Es la generalización por inclusión de una clase a otra más general que la contiene. Por ejemplo, la clase de objeto PERSONA puede verse o definirse como una generalización de las clases ESTUDIANTE y MAESTRO.

La agregación y la generalización están relacionadas con el concepto de jerarquía entre tipos o clases. Dos jerarquías muy conocidas son IS-A y PART-OF. El concepto de IS-A se refiere a la generalización de una clase a otra, así se dice por ejemplo que ESTUDIANTE IS-A PERSONA. El concepto de PART-OF se refiere al hecho de que una clase de objeto (agregado) es una parte de otra clase de objeto (por ejemplo, SECCION es PART-OF de PLANTA-INDUSTRIAL).

I.4.2 MODELO RELACIONAL EXTENDIDO (RM/T).

Este modelo es una propuesta hecha por Codd [CODD 79] en la que se plantea solucionar algunas de las limitaciones semánticas del modelo relacional (las siglas RM/T corresponden a las iniciales de Relational Model Tasmania [15]), por lo que RM/T puede considerarse como una extensión de dicho modelo.

Puesto que RM/T es consistente con el modelo relacional original, las principales características de este último son también características de RM/T como lo es la suposición básica de que mediante el concepto de relación (fundamental en el modelo) puede ser modelando el mundo real.

En cuanto a las diferencias, comenzaremos por subrayar una muy importante, la introducción del concepto de [16]

identificador generado por el sistema (surrogates) .

La función del concepto de 'llave' (que habíamos apuntado que es la de identificar de manera única a los objetos de la BD) queda ahora bajo responsabilidad del identificador generado por el sistema. La ventaja del manejo de la base de datos usando estos identificadores es que el usuario no interviene en el control de ellos (no los conoce), lo cual permite un funcionamiento más efectivo del sistema (17).

Recordemos ahora que una de las principales deficiencias del modelo relacional (señalada en I.2.1) es la no distinción de las relaciones que representan objetos o entidades de las relaciones que representan asociaciones entre dichas entidades. En RM/T se supera este problema estableciendo una clasificación de las diferentes relaciones que podrían existir, de acuerdo a su significado. Así por ejemplo, pueden distinguirse varios tipos de relaciones:

a) Relaciones que representan entidades.
b) Relaciones que representan propiedades de entidades.
c) Relaciones que representan asociaciones entre entidades.
d) Relaciones (llamadas relaciones gráficas y que se incluyen en un catálogo) que establecen la estructura del modelo y proporcionan la facilidad de definir en el Esquema propiedades semánticas a

través de relaciones para

- Agregación cartesiana.
- Generalización.
- Presedencia de eventos.
- Agregación heterogenea.

Esta clasificación de las relaciones lleva asociadas reglas de integridad que por supuesto son más específicas que las reglas de integridad del modelo relacional (por ejemplo, limitan la posibilidad de establecer asociaciones sólo al caso de que los objetos a asociarse ya existan; otro ejemplo de limitación establecida por una regla de integridad sería en el caso de borrado de un objeto, el cual no debe ser posible si éste se encuentra asociado a otro, etc.)

El punto d) involucra aspectos conceptuales muy interesantes que tienen que ver con el manejo y representación de la estructura de objetos complejos (como ya lo hemos dicho, este aspecto nos interesa particularmente).

Codd dice que la agregación cartesiana es una dimensión importante para formar "unidades significativas mayores". Al concepto de agregación de Smith, Codd le llama "agregación cartesiana" para distinguirlo de otro tipo de agregación en RM/T, la que aquí hemos mencionado en el punto d) como "agragación heterogenea" (Codd le llama "cover aggregation").

En RM/T se distinguen tres tipos de agregación cartesiana:

1) Agregación de propiedades a entidades.

2) Agregación de entidades a entidades.

3) Agregación de asociaciones a asociaciones.

La agregación cartesiana permite la creación de objetos complejos, sin embargo dichos objetos están limitados por las restricciones especiales inducidas por las relaciones que permiten dicha agregación.

La agregación heterogenea permite la representación de objetos complejos cuyos constituyentes pueden ser de clases heterogeneas.

La generalización por membrecía (de instancia a clase) ya estaba representada en el modelo relacional (en las entidades). Sin embargo, la generalización por inclusión (de clase a clase) no se facilita sino hasta RM/T, donde se le introduce con el nombre de "generalización incondicional".

Codd propone aumentar la noción de generalización considerando la posibilidad de que una clase de objeto pueda ser generalizada a varias clases, a lo que le da el nombre de "generalización alternativa".

Sólo nos resta mencionar un punto en relación con el Catálogo y es lo relativo a los llamados "eventos". En RM/T se propone un mecanismo de control de eventos (entidades en cuya descripción tiene importancia el tiempo de ocurrencia, de inicio o final), para lo cual se definen relaciones gráficas para precedencia incondicional y precedencia alternativa.

Concluímos esta sección anotando que en RM/T se incluyen las mismas operaciones del modelo relacional, aunque las reglas de inserción, actualización y borrado cambian y se incluyen operadores adicionales.

I.4.3 MODELO FUNCIONAL (DAPLEX).

El modelo funcional fue introducido por vez primera en 1977 [18], aunque con anterioridad ya se habían publicado trabajos donde se plantea la idea de considerar a los sistemas de información como colecciones de funciones.

Una de las razones que hacen a este modelo poderoso (y por lo cual se ha hecho popular) es el hecho de que, dadas sus características, es posible definir un lenguaje asociado al modelo que sea sumamente intuitivo para el usuario, lo cual es natural puesto que las respuestas a las preguntas planteadas en una BD pueden considerarse valores de funciones aplicadas a los argumentos correspondientes.

En 1981 se propone un lenguaje de definición y manipulación de datos llamado DAPLEX [SHIP 81] que trata de explotar todas las posibilidades del modelo funcional. Este trabajo tuvo gran aceptación, y en la actualidad es frecuente que en los textos se identifique al modelo funcional con DAPLEX (en lo que resta de la sección hablaremos indistintamente del modelo funcional y DAPLEX).

Aunque el modelo funcional no fue desarrollado teniendo en mente facilitar el manejo de objetos complejos,

el resultado fue un modelo poderoso en este aspecto. A continuación mencionamos algunas de las principales características del modelo.

Los datos consisten de un conjunto de entidades, y las asociaciones entre ellos se representan mediante funciones que relacionan entidades (se trata de modelar objetos y sus propiedades).

En DAPLEX se proporciona un mecanismo de generalización a través de la capacidad de expresión de tipos de jerarquías entre entidades. Se define un tipo básico ENTIDAD, además de proporcionar inicialmente tipos que se distinguen por tener una representación lexicográfica (por ejemplo STRING). Todas las entidades son subtipos del tipo ENTIDAD, y a su vez las entidades tienen tipos que son estructurados en un jerarquía de tipos (los tipos de entidades son definidos como funciones que no toman argumentos, las propiedades o atributos de las entidades se definen como una colección de funciones).

Las funciones en DAPLEX pueden no tener argumentos (entidades), o tener uno o tantos argumentos como se requieran. La aplicación de las funciones puede regresar un valor o un conjunto de entidades, es decir, el modelo admite funciones multivaluadas. Además las funciones pueden ser aplicadas a uno o varios argumentos, lo que proporciona "un medio conveniente para establecer relaciones que involucran más de dos entidades sin introducir entidades artificiales" [ATKI 84].

Un concepto de particular importancia en el modelo

es el de funciones 'derivadas', y precisamente el reconocer la potencialidad de este tipo de funciones es uno de los métodos de DAPLEX. En el modelo funcional los "datos derivados" por medio de definiciones de funciones derivadas se refieren a nuevas propiedades de objetos basadas en los valores de otras propiedades, es decir, DAPLEX permite la obtención de propiedades de un objeto que son derivados de las propiedades de otros objetos relacionados con él, lo cual implica que el modelo posea una habilidad de inferencia muy interesante (las funciones derivadas no necesariamente deben ser predefinidas). El hecho básico que permite trabajar con funciones derivasdas es la regla de composición de las funciones que, como se sabe, permite aplicar en cadena varias funciones siempre que las mismas tengan los argumentos adecuados.

Las funciones derivados permiten la simplificación de las preguntas requeridas por el usuario y facilitan la representación de relaciones entre entidades directamente mediante la definición de ellas en términos de las relaciones existentes. Además, según Shipman, las funciones derivadas permiten que la semántica de las aplicaciones sea codificada dentro de la descripcion de datos, permitiendo, por lo tanto, expresar las preguntas directamente en términos de esa semántica. Pude decirse también que las funciones derivadas sirven para definir atributos agregados.

Considerando que la modelación de objetos y sus relaciones varía de acuerdo al tipo de aplicación que desea cada usuario, DAPLEX prevee la construcción de vistas del

usuario de la base de datos, las cuales son implementadas por medio de funciones derivadas, lo que permite complejas interrelaciones entre las vistas.

Aunque el diseñador de DAPLEX no expone ampliamente lo referente a las operaciones en el modelo, si señala que los operadores de propósito general de un lenguaje de alto nivel otorgan a DAPLEX un alto poder computacional.

Concluimos esta sección resumiendo las principales características del modelo en cuanto al manejo de objetos complejos. DAPLEX es un modelo sumamente flexible por la facilidad que proporciona para definir relaciones en forma procedural (lo que permite un modelamiento uniforme de todo tipo de relaciones entre objetos) y por la variedad de vistas que soporta. Evita la creación de entidades artificiales para modelar relaciones muchos a muchos. La especificación de jerarquías de tipos (subtipos y supertipos) captura un aspecto importante del modelado semántico de objetos.

Sin embargo, DAPLEX no proporciona un mecanismo para asociar restricciones a las interrelaciones de los objetos, lo que no permite expresar adecuadamente a objetos estructurados complejos.

Otro problema, según se señala en [DAYA 85] es que "mientras que DAPLEX soporta múltiples niveles de abstracción a través de su mecanismo de vistas, falta la habilidad para empacar el mapeo de operaciones desde un nivel de abstracción a otro".

I.4.4 MODELOS DE REDES SEMANTICAS.

Los modelos de redes semánticas fueron desarrollados inicialmente por investigadores del área de inteligencia artificial. Estos primeros trabajos buscaban un objetivo muy preciso (y al mismo tiempo difícil y complejo): la comprensión automática del lenguaje natural, lo cual implicaba avocarse a la tarea de modelar el proceso mediante el cual se llevan a cabo las asociaciones de la memoria humana (19).

Posteriormente, los objetivos del desarrollo de estos modelos se han generalizado y ampliado, por lo cual puede hablarse precisamente de varios modelos (y no de un sólo modelo de redes semántico), los cuales pueden diferir mucho unos de otros. Esto ha sido posible debido a que la caracterización básica de los modelos es muy general (muchos modelos la pueden adoptar), se refiere a la estructura gráfica de los modelos.

La estructura básica de los modelos de redes semánticas es la gráfica (al igual que en el modelo de redes) donde los nodos representan objetos o datos y los arcos representan las asociaciones entre objetos. Sin embargo, la diferencia fundamental con el modelo de redes es que en éste no hay nada intrínseco a la gráfica que sea semántico y en cambio, en los modelos que estamos tratando, debe quedar implícito qué es lo que los nodos y arcos significan y cómo pueden utilizarse.

De acuerdo a la caracterización muy general que se

acaba de dar, puede considerarse que DAFLEX es un modelo de redes semánticas [20], lo mismo que el modelo orientado a objetos (que es el modelo central en esta tesis y que se verá en el siguiente capítulo).

A continuación se mencionan las principales características que presentan en la actualidad algunos de los modelos de redes semáticos desarrollados en el área de bases de datos [21]:

- Diferenciación estricta entre instancia y clase de objeto (es decir, una cosa es el valor del dato y otra su tipo). Se dice que los modelos de redes semánticas fueron los primeros que hicieron esta distinción con claridad.

- Diferenciación estricta entre generalización de instancia a clase y de clase a clase.

- Introducción de mecanismos de generalización y agregación con jerarquías asociadas (las jerarquías IS-A y PART-OF fueron adoptadas por vez primera en modelos de redes semánticas).

- Especificación precisa de reglas de herencia para clases de objetos asociados.

- Introducción del concepto de rol. Este concepto presenta muchas variaciones, según el modelo de redes semánticas específico de que se trate, al-

gunos modelos lo relacionan con el concepto de entidad, otros con ciertas asociaciones y otros distinguen diferentes tipos de roles (en la sección siguiente se hablará con más precisión de la modalidad que adoptará en adelante el concepto de rol en lo que sigue de este trabajo).

- Representación (en muchas posibles formas, según el modelo) de restricciones como parte integral de la definición del esquema, intrínsecamente a la estructura del modelo.

- Posibilidad de manejo de subesquemas restringidos a ciertas instancias. En la mayoría de los modelos esto no es posible ya que un subesquema se define como un subconjunto del esquema (es decir, se define en términos de clases de objetos y no de instancias de las mismas). Sin embargo, algunas aplicaciones requieren el manejo exclusivo de alguna instancia, un ejemplo es el caso de un determinado departamento y sus empleados asociados, la noción de subesquema en los otros modelos sólo permitiría la especificación de un subesquema con todos los departamentos y sus empleados.

Las anteriores características son una muestra de la gran riqueza conceptual que pueden proporcionar estos modelos, aunque algunos autores consideran que hay casos en que se incurre en un exceso de complejidad.

Las grandes ventajas de este tipo de modelos son las facilidades que proporciona para el manejo de objetos y situaciones complejas con una gran expresividad semántica, y su flexibilidad , la cual permite incluir toda una variedad de nuevas ideas.

Por último sólo nos resta anotar que las aportaciones de los modelos de redes semánticas pueden ser aprovechadas en forma independiente del modelo, inclusive en relación con otros modelos de datos.

CAPITULO II.

MODELO DE DATOS ORIENTADO A OBJETOS.

En este capitulo se dará una descripción informal del modelo de datos orientado a objetos[22] que se especificará formalmente en los capitulos IV, V y VI. Este es un modelo que recoge algunas de las aportaciones más interesantes de los modelos semánticos, se habia anotado ya que incluso puede considerársele como un modelo de redes semántico, según la caracterización que se dió en I.4.4.

Este modelo no convencional se ha desarrollado muy recientemente tratando de responder a las necesidades que plantean los sistemas de información modernos (en particular CAD) y, a diferencia de otros modelos no tradicionales (como el funcional) el desarrollo de este modelo desde un principio ha tomado conciencia plena de la necesidad e importancia del manejo de objetos complejos con la semántica asociada a ellos como una parte fundamental (consultar [BUCH 84b], [BUCH 85] y [DAYA 85] principalmente).

Otro aspecto interesante del modelo es que trata de establecer una relación explicita entre los conceptos de sistemas orientados a objetos y los conceptos de bases de datos[23], de hecho, el modelo orientado a objetos po-

ría ser visto como un sistema orientado a objetos, ya que adopta algunas de sus características más importantes. En la sección siguiente trataremos brevemente este aspecto (en lo que sigue, llamaremos en ocasiones simplemente Modelo al modelo orientado a objetos).

II.1 SISTEMAS ORIENTADOS A OBJETOS.

En la actualidad el concepto "orientado a objetos" ha alcanzado una importancia muy especial. Se habla de programación, lenguajes, sistemas, metodologías y arquitecturas orientadas a objetos. Observamos un auge tal del concepto que se dice que en la década de los 80's todo mundo estará a favor de él, " todo manufacturero promoverá que sus productos lo soporten, todo programador lo practicará y nadie conocerá exactamente lo que es " [RENT 82].

Hay que tomar en cuenta que existe una diversidad de opiniones en cuanto lo que significa exactamente "orientado a objetos"[24], no obstante, hay algunos elementos básicos que caracterizan a un sistema orientado a objetos (en los que casi todos coinciden) y que el modelo que tratamos en este capítulo satisface.

El concepto básico es el de objeto. En este trabajo se ha manejado ya este concepto en una forma intuitiva (se introdujo en la sección I.3.1). Un objeto no puede definirse en forma intrínseca, sino que debe ser visto desde el exterior y caracterizado por sus propiedades[25].

En un sistema orientado a objetos todos los datos son vistos indistintamente como objetos. El ejemplo de CAD es ilustrativo ya que es factible imaginar como en un sistema de este tipo todo nuestros datos pueden ser organizados como objetos (dado que el sistema se ocupa precisamente del diseño de ciertos objetos).

Un concepto estrechamente ligado al de objeto es el concepto de "clase" (26). La clase de un objeto es la unidad estructural que lo define o describe, por lo tanto, no puede existir ningún objeto en el sistema que no tenga una clase asociada a él, es decir, que no sea de una clase determinada previamente definida. A una clase de objeto se acostumbra llamarle simplemente "clase", análogamente, a una instancia de una clase de objeto especifica se le acostumbra llamar simplemente "objeto".

Decíamos que en un sistema orientado a objetos todos los datos son vistos uniformemente como objetos ya que:

- Todos los datos en él son objetos, ningún item puede pertenecer al sistema si no es un objeto.
- Todos los objetos son indiferenciables entre si, es decir, podrán pertenecer a diferentes clases pero el sistema los podrá manejar siempre como un todo, indistintamente de la clase, a través de su identificador interno.
- Todas las clases de objetos son uniformes considerando el hecho de que no se distinguen desde

el sistema clases especiales (no existe la noción de niveles o tipos que diferencien una clase de otra).

Un sistema orientado a objetos requiere la capacidad de localizar y alojar a las clases y a todos los objetos del ambiente, además de tener la capacidad de recuperar la información de dichos objetos. También debe ser capaz de crear clases nuevas e insertar objetos.

Otro elemento muy importante en este tipo de sistemas - que se considera primordial - es la capacidad para facilitar la herencia de propiedades de un objeto a otro, lo cual permita compartir la mayor cantidad de información posible.

En los artículos del área se mencionan las siguientes estructuras relacionadas con la herencia:

- Subclase. Es una especialización de una clase existente.

- Superclase. Es una clase (ya existente) la cual tiene subclases que son especialización de ella.

Hay otros atributos de sistemas orientados a objetos que podrían estudiarse, sin embargo, las características que se han presentado son las que tienen mayor relevancia en el Modelo y serán expuestas con mayor profundidad más adelante.

Para terminar con esta sección se mencionan algunas de las ventajas que los sistemas orientados a objetos proporcionan y que ilustran su importancia en la actualidad.

Los sistemas orientados a objetos facilitan el diseño y el mantenimiento del software proporcionando herramientas conceptuales más acordes con la semántica de las aplicaciones. En el aspecto técnico ofrecen tiempos de desarrollo más corto, un alto grado de optimización del aprovechamiento del espacio (código compartido) y una gran flexibilidad.

Aunque las ventajas prácticas son importantes, también se aducen otro tipo de argumentos a favor de estos sistemas. Se dice que ellos permitirán un cambio en los patrones de pensamiento existente (en cuanto a modelación de datos o programación, por ejemplo), que evite la fragmentación de la información, lo cual se practica en los sistemas tradicionales pues esta fragmentación es favorecida por la estructura de dichos sistemas.

II.2 DESCRIPCION INFORMAL DEL MODELO.

La descripción informal del modelo orientado a objetos que se presenta en esta sección introduce una visión global preliminar de estructuras y conceptos básicos. El objeto de esta exposición es proporcionar al lector una idea intuitiva previa del Modelo. Debe tenerse en cuenta

que los conceptos no quedarán precisamente definidos hasta el capitulo IV donde el Modelo es especificado formalmente.

Se ha afirmado ya que el Modelo puede caracterizarse como un modelo de redes semánticas en tanto que satisface la condición básica más general de ellos: su estructura abstracta corresponde a la de una gráfica donde los nodos representan objetos y donde los arcos representan las asociaciones entre los objetos. En los sucesivo daremos el nombre de roles a dichas asociaciones.

El modelo también está caracterizado por establecer una diferenciación estricta entre instancias y clases de objeto. Asumiremos para representar esto las siguientes convenciones:

- El simbolo de un nodo de la gráfica será un círculo, éste representará una clase de objeto.
- Un asterisco dentro del círculo representará una instancia de objeto cuya clase es la del círculo que lo contiene.
- Una flecha o recta de círculo a círculo simbolizará la existencia de un rol entre las clases de objetos asociadas.
- Una recta de asterisco a asterisco simbolizará la existencia de un rol entre instancias de objetos asociados (presuponiendo siempre que

existe el rol correspondiente entre las clases de los objetos).

- Cuando sea necesario representar atributos de una clase de objeto lo haremos mediante un rectángulo unido al círculo que representa la clase.

El modelo puede ser considerado un sistema orientado a objetos en el cual toda la información está organizada en objetos, por esto, el concepto de rol no corresponde al de una entidad independiente del objeto sino que todo rol está definido intrinsecamente en cada clase de objeto. ¿Cómo se establece esto?. A continuación se expone con detalle la constitución estructural de un objeto para contestar a esa pregunta y a otras más como: ¿Los atributos son entidades que pueden considerarse como objetos por sí mismos?, cuando se tiene un objeto complejo ¿cómo se representa éste y sus objetos constituyentes?, etc.

II.2.1 ESTRUCTURA CONCEPTUAL DE UNA CLASE DE OBJETO.

La estructura de cada objeto está determinada por la definición de la clase de objeto respectiva. Cada clase está formada por tres elementos conceptuales básicos:

- Un conjunto de objetos agregados.
- Un conjunto de roles a través de los cuales se establece la asociación entre la clase y sus objetos agregados.

- Un conjunto de atributos que califican a la clase, cuyos elementos son las propiedades características de la clase.

Todas las clases, todos los roles y todos los atributos tienen un nombre.

II.2.1.1 Identificadores de instancia.

A cada instancia de una clase de objeto se le asocia un identificador de instancia único.

Dicho identificador de instancia único corresponde al concepto de "identificador generado por el sistema" de RM/T, por lo cual el concepto tradicional de "llave" no aparece como tal en este Modelo. Ya hemos señalado la ventajas de utilizar identificadores internos - a los cuales no tiene acceso el usuario - en la mayor efectividad del sistema (consultar nota (17)).

II.2.1.2 Objetos agregados.

En la definición de una clase, el conjunto de objetos agregados está formado por los nombres de las clases de objetos que pueden ser constituyentes de la clase definida (a veces les llamamos sencillamente "agregados").

Cuando el conjunto de objetos agregados es diferente

de vacio se tiene un objeto complejo, cuando esto sucede, necesariamente el conjunto de roles es diferente también de vacio. Puesto que en la definición del Modelo no se distingue entre un objeto complejo y uno que no lo es (es decir, el objeto cuyo conjunto de agregados es vacío)[28], los elementos del conjunto de objetos agregados pueden o no ser complejos indistintamente.

Hay que subrayar que la definición de clases de objetos agregados siempre se hace especificando a través de qué rol se está agregando. Esto será mejor comprendido en una siguiente sección, donde se analiza el concepto de agregación subyacente en el Modelo.

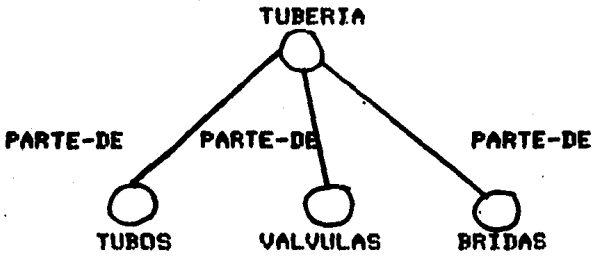
II.2.1.3 Roles.

El conjunto de roles de una clase merece una atención especial, pues puede decirse que el concepto de rol es uno de los que encierra mayor riqueza semántica en el Modelo.

A través de los roles no solo se establecen asociaciones entre objetos sino que se definen diferentes tipos de jerarquías , reglas de herencia y restricciones o condiciones sobre las posibles relaciones, todo esto con una gran flexibilidad.

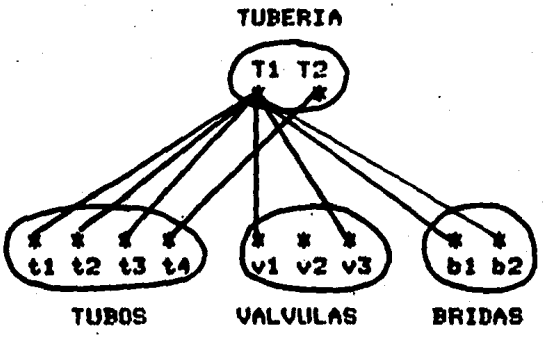
Cada rol de una clase define una asociación entre instancias de objetos de esa clase e instancias de un conjunto de clases. Por ejemplo, si tenemos un objeto

de clase TUBERIA cuyos objetos constituyentes son objetos de las clases TUBOS, VALVULAS y BRIDAS, entonces, en la clase TUBERIA debe definirse un rol que asocie TUBERIA con sus partes, a dicho rol podría llamársele PARTE-DE (el nombre del rol es un auxiliar semántico para tipificar la asociación).



La gráfica anterior indica que la clase de objeto TUBERIA tiene como objetos agregados a las clases TUBOS, VALVULAS y BRIDAS a través del rol PARTE-DE.

Una vez definida la clase TUBERIA, podrían tenerse instancias de objetos asociadas según se ilustra en la gráfica siguiente.

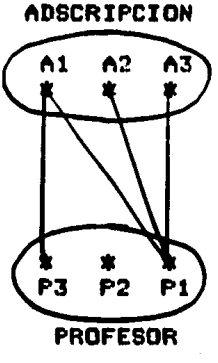


Esta gráfica nos indica que las partes de la tubería T1 son tres tubos (t1,t2,t3), dos válvulas (v1,v3) y dos bridas (b1,b2) y que las partes de la tubería T2 son un

tubo (t4), ninguna válvula y ninguna brida.

Al concepto correspondiente a "objetos agregados" a nivel de instancias le damos en ocasiones el nombre de "hijos" de una clase a través de un rol dado. A las instancias de donde parte el rol les llamamos "padres". Por ejemplo, en la gráfica anterior, t1, t2 y t3 son "hijos" de la clase TUBOS a través del rol PARTE-DE de T1; T2 es padre a través del rol PARTE-DE de la instancia t4 de TUBO.

Un ejemplo donde hay varios "padres" es el caso de la clase de objeto ADSCRIPCION con un agregado PROFESOR a través del rol TRABAJA-EN, con las siguientes asociaciones de instancias.



Aquí, P1 tiene como "padres" a A1, A2, y A3 (esta situación puede presentarse en varios casos, por ejemplo en la UNAM es posible que un profesor esté adscrito a varias dependencias o facultades).

Aprovecharemos este último ejemplo para señalar que en este Modelo son muy fácilmente expresables las asociaciones o relaciones de "muchos a muchos".

Otra observación es que debe notarse que una clase puede tener un conjunto de roles definidos, y que cada rol puede asociar un conjunto de clases de objetos. Tendrían que darse muchísimos ejemplos más para ilustrar los diferentes casos que pueden presentarse y aún así no quedarían agotadas todas las posibilidades, sin embargo, después de la especificación formal del Modelo se tendrá una visión completa de los casos que admite.

Condicionamiento de roles.

Existe la posibilidad de condicionar a los roles, es decir, indicar restricciones respecto a algunas instancias o atributos especiales. La capacidad de condicionar a los roles es uno de los aspectos más interesantes del Modelo. Dichas condiciones (a las que también llamaremos condiciones de agregación) se establecen mediante expresiones booleanas de un lenguaje especialmente definido para este propósito (29).

Será hasta el capítulo V donde se expondrá con detalle cómo es posible establecer las condiciones sobre los roles, en dicho capítulo definiremos un condicionamiento de roles al cual le llamaremos condicionamiento sobre atributos.

El condicionamiento de roles sobre atributos permite restringir asociaciones entre instancias determinadas. Este tipo de condiciones establecen algunas características

especiales que los atributos de las instancias deben satisfacer. Si la condición no se cumple, entonces la asociación de las instancias no se efectúa.

Por ejemplo, supongamos que la clase TUBOS tiene entre sus atributos a DIAMETRO, y que la clase TUBERIA , también tiene un atributo que se refiere al diámetro y que se llama asi mismo DIAMETRO (el nombre podría ser diferente), entonces, mediante la definición de una condición del rol PARTE-DE, podemos establecer que solo se asocien instancias de TUBERIA a instancias de TUBOS que tengan el mismo diámetro.

El ejemplo anterior es simple, considerando que pueden plantearse en el Modelo condiciones mucho más complejas pero sirve como un primer acercamiento al concepto.

II.2.1.4 Atributos.

Otro concepto muy importante para la capacidad expresiva del Modelo, mediante el cual sería potencialemente posible definir prácticamente todas las modalidades de herencia que fueran necesarias es el concepto de atributos.

Se indicó ya, en forma general, que la definición de una clase de objeto incluye la definición de un conjunto de atributos que corresponden a la clase y la caracterizan. Existen en el Modelo cuatro categorías de atributos claramente diferenciadas:

- Atributos primitivos.

- Atributos generados.
- Atributos heredados.
- Atributos retribuidos.

Todos los atributos, de cualquier categoría, forman parte integral de cada clase, es decir, no pueden existir en forma independiente de la clase.

La categoría más sencilla es la de los atributos primitivos, éstos se definen simplemente asociando a cada nombre de atributo su dominio de posible valores.

Los atributos generados, heredados y retribuidos son categorías más elaboradas que permiten el establecimiento de jerarquías y reglas de herencia. Para la definición de estos atributos se utilizan fórmulas que indican cómo ha de llevarse a cabo el cálculo del valor en cuestión. Dichas fórmulas deben ser expresiones de un lenguaje diseñado especificamente para este propósito.

En el capitulo V se verá como cada expresión del lenguaje tendrá variables que hagan referencia a la clase de objeto y a los atributos correspondientes que habrán de proporcionar los valores necesarios para el cómputo del atributo definido. En seguida veremos una introducción a las diferentes categorías, ejemplificando qué tipo de fórmulas son las que se utilizan para su definición, para lo cual supondremos provisionalmente que contamos con un lenguaje parecido al que nos permite la formulación de ecuaciones.

Los atributos generados son atributos que pueden ser

generados, como su nombre lo indica, por otros atributos del mismo objeto. La necesidad de estos atributos proviene del hecho que se presenta en algunas clases de objetos cuando algun (o algunos) atributo depende de los valores de otros atributos de la misma clase.

Por ejemplo, supongamos una clase donde se han definido dos atributos X, Y cualesquiera, entonces, si se tiene otro atributo, digamos Z, que depende de los valores de X y Y, dicho atributo deberá definirse como atributo generado, lo cual se hará asignándole una expresión del estilo siguiente:

Y + X

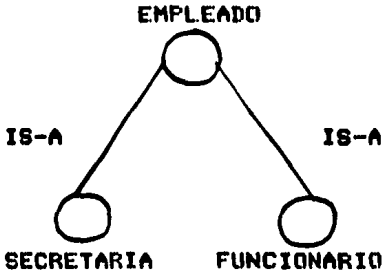
En este caso, el valor de Z sería igual a la suma de los valores de Y y X.

Los atributos heredados son atributos que una clase de objeto puede heredar de otra clase. Para que la herencia pueda darse, la clase del objeto que va a heredar debe ser un agregado de la clase de objeto de la que se va a heredar algún o algunos atributos.

Hay una gran variedad de posibles formas en que se pueden heredar atributos a una clase: desde diferentes objetos "padre", desde varios de sus atributos, a través de roles de agregación distintos, etc. En seguida, se ejemplifica con un caso simple el concepto, otros ejemplos se presentarán en la especificación formal.

Supongamos que se ha definido la clase de objeto EMPLEADO y sus clases de objetos agregados SECRETARIA y

FUNCIONARIO a través del rol IS-A, como se muestra en la gráfica.



Si la clase EMPLEADO tiene como atributos primitivos a RFC (Registro Federal de Causantes), SUELDO, HORA-ENTRADA y HORA-SALIDA entonces podrían heredarse los atributos RFC y SUELDO a ambos agregados y además podría desearse heredar HORA-ENTRADA y HORA-SALIDA sólo a la clase SECRETARIA (suponiendo que en un empleado de clase FUNCIONARIO esos atributos no son necesarios) ¿ Cómo podría establecerse la herencia?

El problema se resuelve en la forma siguiente: en la clase SECRETARIA se definen como atributos heredados a los atributos RFC, SUELDO, HORA-ENTRADA, HORA-SALIDA. Note que estos atributos pueden tener el mismo nombre que los atributos primitivos de EMPLEADO, esto es posible ya que en el modelo formal siempre se hará referencia a nombres de atributos en relación con su clase por lo cual no habrá motivo de confusión ni necesidad de inventar nombres diferentes que en realidad corresponden al mismo concepto. El lector puede hacerse una idea de como se define el atributo heredado RFC de la clase SECRETARIA, con la siguiente fórmula.

SECRETARIA.RFC !--> EMPLEADO.RFC

Esta fórmula significa que al atributo RFC de SECRETARIA se le asocia el valor del atributo RFC del objeto de EMPLEADO.

De forma análoga se definirán los demás atributos heredados de SECRETARIA. Para el caso de FUNCIONARIO, se definen como atributos heredados solo a RFC, SUELDO, así:

FUNCIONARIO.RFC !--> EMPLEADO.RFC

FUNCIONARIO.SUELDO !--> EMPLEADO.SUELDO

La primera fórmula significa que el el atributo RFC de FUNCIONARIO hereda el valor del atributo RFC de EMPLEADO. La segunda fórmula significa que el atributo SUELDO de FUNCIONARIO hereda el valor del atributo SUELDO del objeto EMPLEADO.

El ejemplo anterior muestra como puede darse herencia simple, transmitiendo un valor de atributo a otro, sin embargo, la herencia puede darse operando en otras formas los valores heredados, dependiendo de las capacidades operativas que proporcione el lenguaje de expresiones.

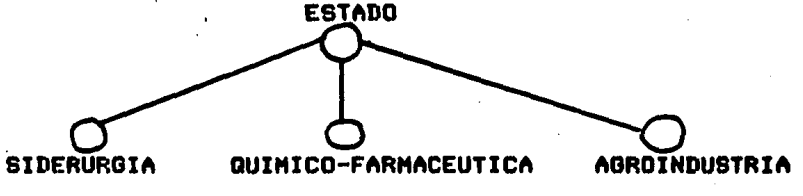
Se ha ilustrado de manera informal como los nombres de las variables de las expresiones del lenguaje que sirve para definir a los atributos heredados deben formarse de dos partes: el nombre de la clase (de la cual se hereda) y el nombre del atributo de dicha clase que se quiere heredar. Sin embargo, hay otro elemento que debe hacerse

explícito en los nombres de las variables y es el nombre del rol de agregación específico a través del cual va a heredarse. En el ejemplo anterior no se indica a través de cual rol se hereda pues el caso expuesto es sencillo y solo se tiene un rol de agregación, no obstante, en general debe indicarse en los nombres de las variables a través de que rol se está accesando la información.

Puede advertirse que el manejo de la herencia que proporciona el Modelo a través de sus atributos heredados es sumamente flexible y no se encuentra limitado a adoptar formas predefinidas, estáticas y rígidas de herencia sino que es adaptable a diversas situaciones que se presentan en aplicaciones de bases de datos.

El concepto de atributos retribuidos es un concepto poco trabajado en otros modelos. Informalmente hablando, un atributo retribuido es un atributo de una clase de objeto cuyo valor está determinado por uno o varios atributos de sus objetos agregados.

Pongamos el siguiente ejemplo: dada una clase de objeto ESTADO cuyos agregados definidos a través del rol RAMA-DE-ACTIVIDAD son las clases SIDERURGIA, QUIMICO-FARMACEUTICA y AGROINDUSTRIA, con la siguiente gráfica.



Supongamos que cada uno de sus agregados tiene un atributo llamado INDICE que corresponde al índice de

crecimiento de la rama de actividad.

Si suponemos además que es deseable contar con un índice de crecimiento económico global del estado definido por el promedio de los índices de crecimiento de cada rama de actividad, entonces esta situación puede plasmarse en el Modelo definiendo un atributo retribuido en la clase ESTADO, que podríamos llamar INDICE-GLOBAL y el cual se definiría de una manera similar a esta:

```
ESTADO.INDICE-GLOBAL !--> ( (SIDERURGIA.INDICE +
                             QUIMICO-FARMACEUTICA.INDICE +
                             AGROINDUSTRIA.INDICE) / 3 )
```

En este ejemplo, los nombres de las variables que definen al atributo INDICE-GLOBAL no hacen referencia al rol de agregación a través del cual se retribuye, pero en general sí se deberá indicar el rol como veremos en la especificación formal.

Las categorías de atributos que se han presentado dan una idea de las posibilidades del Modelo pero, repetimos, será hasta el capítulo V donde se pueda comprender mejor toda la capacidad expresiva que puede obtenerse.

Además de las categorías descritas, podrían definirse algunas otras que se consideraran necesarias e inclusive el lenguaje de expresiones que sirve para la definición de los atributos pudiera ser extendido aumentando también en esa forma las capacidades del Modelo.

II.2.2 OPERACIONES.

Hemos visto ya dos de los elementos básicos que caracterizan al Modelo: la forma en que se organizan los datos (en nuestro caso es mediante objetos) y la forma en que se asocian estos datos (en este modelo es a través de roles), solo nos falta ahora referirnos a la forma en que se llevan a cabo las operaciones sobre los datos.

Puesto que en el Modelo toda la información se encuentra estructurada en objetos, es lógico inducir que toda operación se efectúa sobre un objeto como entidad indivisible y no hay forma posible de operar sobre partes del objeto (digamos roles o atributos).

Como en todo modelo, las operaciones básicas son inserción, borrado, recuperación y actualización. En lo que sigue se describen solo operaciones sobre objetos relacionadas con la inserción y el borrado, ya que éstas son suficientemente ilustrativas de cómo se opera con objetos en el Modelo.

II.2.2.1 Inserción.

Es la inserción de una instancia de una clase de objeto previamente definida. Es importante observar que el Modelo admite la inserción de instancias "repetidas" de un objeto, es decir, puede haber dos o más instancias iguales

en una misma clase de objeto .

Antes de abordar la presentación del concepto de agregación molecular , discutiremos brevemente las operaciones de desconexión, borrado y algunas operaciones de conexión necesarias para la agregación molecular. Precisamos que el término objeto molecular (o instancia molecular o simplemente molécula) se empleará para hacer referencia a un objeto que es complejo.

II.2.2.2 Desconexión.

Esta operación tiene muchas variantes, dada la variedad de asociaciones posibles de una instancia con otras. Puede desconectarse a una instancia molecular de: otra instancia, todas las instancias de un agregado, todas las instancias de todos los agregados de la molécula, etc.

Para efectuar el borrado de una instancia hay que desconectarla de todos los roles que lleguen o salgan de ella, por lo cual decimos que la desconexión es un requisito previo del borrado.

II.2.2.3 Borrado.

El borrado de una instancia implica forzosamente que

dicha instancia sea desconectada previamente de todos los roles que llegan o salen de ella, si esta condición no se cumple la base de datos sería inconsistente. Ya desconectada la instancia puede procederse a su borrado.

II.2.2.4 Conexión.

La operación de conexión puede tener las siguientes variantes. Conexión de una instancia molecular con:

- Una instancia de uno de sus agregados a través de un rol específico.
- Todas las instancias de un agregado de la molécula a través de un rol.

En todos los casos, la conexión de instancias debe hacerse siempre y cuando no exista un condicionamiento de roles que lo impida.

Otras operaciones de conexión posibles son la conexión de todas las instancias de una molécula con:

- Las instancias de un agregado que satisfagan las condiciones de agregación a través de un rol.
- Las instancias de un agregado a través de todos los roles para las que se cumplan las condiciones de agregación.
- Las instancias de todos los agregados de la molécula a través de todos los roles para las que se cumplan las condiciones de agregación.

El tercer caso de conexión señalado arriba es el más general y es el que corresponde al concepto de agregación molecular. A la discusión de dicho concepto se dedicará la sección siguiente.

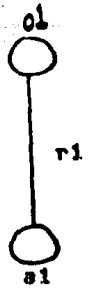
II.2.3 AGREGACION MOLECULAR.

Se ha señalado ya que hay dos tipo de abstracciones de datos , agregación y generalización , que son conceptos importantes incluidos en la mayoría de los modelos semánticos aunque con diferentes acepciones. A continuación se hacen algunas observaciones respecto al concepto en el contexto del modelo de datos orientado a objetos.

El significado del término agregación molecular (31) (o simplemente agregación) ha ido exponiéndose implícita e intuitivamente a lo largo de la descripción del Modelo hecha hasta el momento. Aquí precisaremos algunas ideas.

La agregación es un proceso que tiene que ver con objetos complejos, o como aquí les hemos llamado, con objetos moleculares.

Un objeto complejo o molécula es aquel cuya clase tiene definido un conjunto de agregados con al menos un elemento. Por ejemplo, la clase molecular o1 que se muestra en la gráfica.



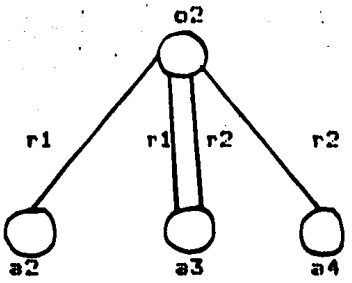
Observamos que la asociación de objetos de clase a1, a objetos de clase o1 forzosamente se hace a través de un rol r1 que los liga.

A esta asociación se le llama agregación, por lo cual a los roles tambien les llamamos roles de agregación.

En todos los casos, los agregados definidos en una clase deben ser clases previamente definidas. Así, en el ejemplo anterior debe suponerse que a1 es una clase ya definida .

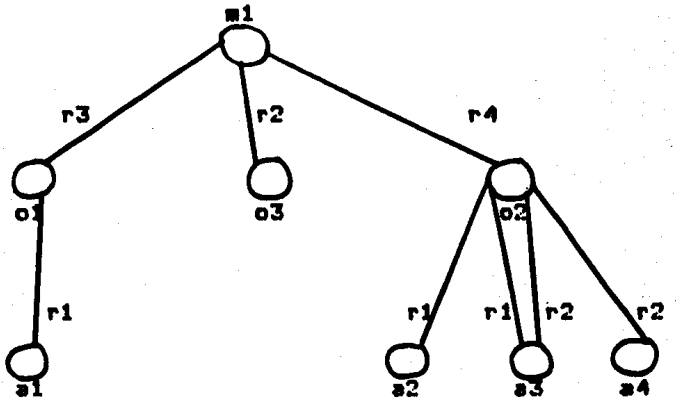
La agregación puede verse como un proceso que genera un objeto complejo independientemente que los objetos agregados sean o no complejos. En el ejemplo anterior o1 es una clase de objetos complejos independientemente de que la clase a1 sea o no de objetos complejos.

Para que la agregación esté bien definida, la definición de la clase o1 debe establecer que a1 es un agregado y que r1 es un rol de o1, en caso de que hubiera más agregados se indicaría qué otras clases de objetos podrían agregarse y a través de qué rol. El siguiente es otro ejemplo con más agregados y roles.



En la definición de la clase o2 no es suficiente indicar que a2, a3 y a4 son agregados, y que r1 y r2 son los roles de o2, sino que es necesario establecer que a través de r1 pueden agregarse a2 y a3, y que a través de r2 se pueden agregar a3 y a4. Este ejemplo nos ilustra además que un agregado puede asociarse a través de tantos roles como sea necesario (es el caso de a3).

En los dos ejemplos anteriores se muestra la agregación de objetos no complejos, en el siguiente ejemplo veremos un caso de agregados complejos. La gráfica de la clase m1 es la siguiente.



En la definición de la clase m1 es necesario estable-

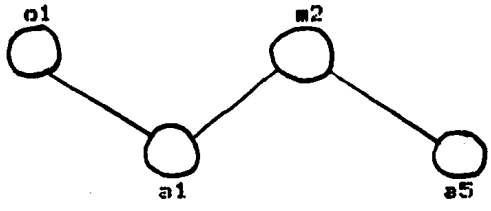
cer que sus agregados son o1, o2 y o3 , que sus roles son r2, r3 y r4, además debe indicarse que a través de r2 puede agregarse o3, que a través de r3 puede agregarse o1 y que a través de r4 puede agregarse o2.

En este ejemplo puede notarse que los nombres de los roles no son exclusivos de una clase particular. En mi se tiene al rol r2 que es un nombre de rol también para la clase o2. Esto tiene la ventaja de que la semántica de las asociaciones puede reflejarse en el nombre del rol más adecuadamente. Por ejemplo, en el caso de un rol tipo IS-A , este nombre puede usarse siempre que se haga referencia al mismo concepto de asociación para cualquier clase.

Se afirma en [BUCH 84b] que no existen SMBD que soporten objetos moleculares disjuntos, no-recursivos, no-disjuntos y recursivos. Todos los ejemplos vistos hasta el momento, han mostrado que el Modelo acepta objetos moleculares disjuntos y no-recursivos, entendiendo por moléculas disjuntas aquellas en que ninguno de sus agregados pertenece al conjunto de agregados de otra molécula, y entendiendo por moléculas no-recursivas aquellas en que ninguno de sus agregados puede ser el mismo objeto molecular.

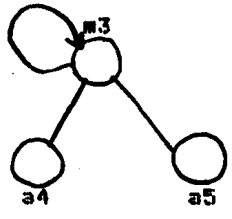
Dado que en el Modelo no se establecen limitaciones respecto a las clases de objetos que pueden pertenecer a los conjuntos de agregados de una molécula (siempre y cuando sean clases definidas), podemos afirmar que este Modelo también acepta a los objetos moleculares no-disjuntos y recursivos. Una molécula es no-disjunta cuando al

menos uno de sus agregados es también agregado de otra molécula. Por ejemplo



En este caso las clases moleculares o1 y m2 son no-disjuntas ya que comparten al agregado a1.

Una molécula es recursiva cuando al menos uno de sus agregados es la molécula misma. Ejemplo



La definición de la clase de objeto m3 debe establecer que al conjunto de sus agregados pertenecen las clases a4, a5 y m3.

La otra forma de abstracción, la generalización, también se encuentra presente en el Modelo en sus dos tipos.

La generalización de instancia a clase en este Modelo puede aplicarse a toda instancia ya que toda instancia u objeto debe pertenecer a una clase de objeto que lo generaliza.

La generalización de clase a clase, según se definió en I.4.1 es la generalización, por inclusión, de una clase

a otra más general. Este tipo de generalización se presenta en algunos modelos a través de la jerarquía IS-A que se refiere a la generalización de una clase (por ejemplo ESTUDIANTE) a otra más general (por ejemplo PERSONA). En algunos de esos modelos se establece la herencia automática de todos los atributos de la clase más general (como PERSONA) a las clases incluidas en ella (como ESTUDIANTE).

Hemos visto que en el Modelo es posible establecer este tipo de jerarquía al definir los atributos heredados, siempre y cuando haya un rol de agregación que lo permita. En el ejemplo de las clases ESTUDIANTE y PERSONA, debe definirse un rol de agregación, que podría llamarse IS-A, en PERSONA, y en ESTUDIANTE deben definirse como atributos heredados todos aquellos atributos que se desee heredar de PERSONA. Esta es la forma en que se da la generalización de clase a clase en el Modelo.

Es importante notar que los roles tipo IS-A no están rigidamente predeterminados en el Modelo, sino que pueden existir diversas modalidades de herencia adecuadas para cada caso de agregación, como vimos en los ejemplos que ilustran los atributos heredados. Ahí vimos como en algunos casos pueden heredarse solo algunos de los atributos, no necesariamente todos. Este manejo de la herencia permite modelar más adecuadamente aspectos del mundo real pues propociona mucha flexibilidad y variedad (puede convinarse el manejo de atributos heredados con atributos generados y retribuidos para obtener los re-

sultados deseados) .

La jerarquía PART-OF que en otros modelos es también implementada en forma rígida, aquí puede adquirir las formas que mejor se adapten a las necesidades. Una jerarquía de este tipo puede establecerse en el Modelo definiendo en el conjunto de agregados de una clase a aquellos que sean parte de ella. Al rol a través del cual se agregan puede llamársele PART-OF, PARTE-DE, PARTES, o cualquier otro, de preferencia que refleje el significado de la agregación.

Hemos visto ya como la agregación puede limitarse por medio del condicionamiento de roles. Este mecanismo, como ya vimos, permite aumentar la variedad y flexibilidad de la agregación molecular en el Modelo, además del aumento de su capacidad expresiva. En adelante, algunas veces haremos referencia a las condiciones de agregación de los roles con el término 'restricciones' (33) .

CAPITULO III.

DISCUSION ACERCA DE ESPECIFICACIONES FORMALES

En este capítulo se presentan algunos elementos básicos acerca de especificaciones formales como: qué es una especificación en general, qué es una especificación formal, qué son lenguajes formales, cuales son las ventajas y desventajas de las especificacines formales, etc.

Además, se introduce el lenguaje de especificación llamado "Z" que usaremos para formalizar el modelo orientado a objetos.

III.1 NECESIDAD DE UNA ESPECIFICACION FORMAL.

En términos generales se entiende por especificación de un sistema a la descripción de sus propiedades y características.

La especificación -sea formal o informal- de un sistema de software puede servir para los siguientes propósitos:

- Como base en el diseño del sistema mismo.
- Como modelo experimental de arquitectura, subsistemas, estructuras, etc.
- Como interface entre requerimientos del cliente y diseñador, entre diseñador e implementador o

entre implementador y usuario.

¿ Porqué el empleo de lenguaje natural no resulta adecuado para dar una especificación informal efectiva de un sistema de cómputo ?.

Son varias las razones, la principal es que el lenguaje natural, nos referimos en nuestro caso al español, tiene varias posibles interpretaciones para algunos de sus términos lo cual puede generar ambigüedades; a esto se añade la variedad de palabras que pueden designar un mismo concepto, lo cual aumenta la posibilidad de confusiones y errores en la interpretación de la especificación.

Otra razón es que las especificaciones informales producidas al emplear un lenguaje natural son, por lo general , incompletas, sobre todo en el caso de sistemas complejos y de gran tamaño.

Consideremos por ejemplo la muy frecuente situación a la que nos enfrentamos como usurarios al tratar de descifrar algun aspecto que nos interesa en un manual. Ya sea que dicho aspecto sí se encuentra considerado en él, pero su explicación es oscura, o ya sea que dicho aspecto no aparece en lugar alguno. En cualquier caso, no se cumple de manera efectiva con el propósito de la especificación, en este ejemplo la interface implementador / usurario no se da adecuadamente.

La especificación informal del modelo orientado a objetos presentado en el capitulo anterior, hecha en un lenguaje natural tampoco es una especificación efectiva, aunque se trató de hacer explícita la interpretación para

cada término relevante y se puso cuidado en prevenir al lector cuando se emplearon varios términos para un mismo concepto. El principal problema de esta espeficación informal es que es incompleta y por lo tanto, no es suficiente para abarcar en forma total las posibilidades del Modelo, ni para poder determinar en cualquier caso particular (por ejemplo de agregación) si está permitido o no.

III.1.1 QUE ES UNA ESPECIFICACION FORMAL.

El lenguaje o notación empleada para especificar debe ser tal que no se preste a generar ambigüedades al permitir varias interpretaciones, es decir, su semántica debe ser precisa. Además la notación debe proporcionar formatos cortos y concisos para las descripciones necesarias. Los lenguajes que satisfacen estas características son conocidos como lenguajes formales.

Considerando lo anterior, podríamos preguntarnos si un lenguaje formal cualquiera, como por ejemplo un lenguaje de programación sirve para los propósitos de una especificación. La respuesta es negativa, debido principalmente a dos factores: el principal es que los propósitos de un lenguaje de programación (hacer operar un procesador) no coinciden con los propósitos de una especificación (describir propiedades y características), el otro factor se refiere a la flexibilidad en los tipos

de datos que requiere en general una especificación, lo cual no es proporcionado por los lenguajes de programación pues operan con tipos de datos restringidos (los tipos de datos requeridos en una especificación son los que se adapten mejor al problema en forma natural).

No es suficiente, por lo tanto, especificar usando un lenguaje que sea formal, se requiere la utilización de un lenguaje creado con el propósito de especificar formalmente. A estas notaciones o lenguajes les llamamos lenguajes de especificación formal , las matemáticas son la fuente básica para estas notaciones.

Resulta sensato así el siguiente comentario: "Si la intención es escribir poesía o cartas de amor -use lenguaje natural; si la tarea es instruir a una computadora -use un lenguaje de programación; si el deseo es escribir una especificación -un lenguaje de especificación formal debe ser usado" (34).

Llamamos especificación formal a una especificación escrita utilizando un lenguaje formal. Existen varios lenguajes de este tipo y varias técnicas y métodos de especificación formal, pero las características generales que debe satisfacer una especificación formal de un sistema son: consistencia (no contradicción), inambigüedad y completez. Además debe ser concisa, evitando la inclusión de detalles de implementación y procurando reflejar el comportamiento del sistema en abstracto.

Para esto es fundamental seleccionar estructuras y tipos de datos que correspondan más a las estructuras na-

turales de los objetos que a las necesidades de la implantación.

III.1.2 PROPOSITOS Y VENTAJAS.

Al principio del capítulo señalamos algunos de los propósitos de una especificación en el contexto de sistemas de software. Hemos visto que una especificación informal difícilmente podría satisfacer adecuadamente dichos propósitos, en cambio, una especificación formal además de responder satisfactoriamente a ellos ayuda a evitar problemas muy comunes como son:

- No correspondencia entre requerimientos del cliente respecto a un sistema y el diseño del mismo.
- Errores en la implementación en cuanto a la no correspondencia entre diseño e implantación o entre especificación e implantación (35).
- Desperdicio de recursos al utilizar tiempo de máquina en la experimentación de modelos o sistemas. Una especificación formal previa a la implantación puede funcionar mucho más efectivamente como modelo experimental (algunos autores dan la mayor importancia a este punto).

Otras ventajas que se obtienen con la especificación formal de sistemas se resumen adecuadamente en los siguien-

tes puntos [QUIN 85]:

- La especificación formal de un sistema, como base preliminar en el diseño del mismo da la posibilidad de descubrir ambigüedades, omisiones y contradicciones en los requerimientos iniciales solicitados. Estos problemas a veces son muy difíciles de detectar de otra forma.
- Sin recurrir a la implementación pueden investigarse algunas consecuencias sobre determinadas decisiones de diseño (la práctica común de esperar hasta el momento de implantar para conocer las consecuencias resulta costosa). En base a la especificación pueden sugerirse las modificaciones necesarias antes de que el proyecto se encuentre en un estado posterior de desarrollo.
- Una implementación válida de un subsistema puede ser incorporada en la implementación de un sistema grande con bastante seguridad de que se comportará como se ha especificado y de que produce los resultados esperados.
- Las especificaciones formales son una guía insustituible para la elaboración de todos los manuales y guías de operación necesarios, que resulten comprensibles (si se redactan bién) además de completos y consistentes. Como consecuencia de esto el mantenimiento del sistema se facilitará y su funcionalidad aumentará.

En un articulo Guttag menciona un beneficio más que puede reportar la especificación formal de un sistema:

> " Los conceptos desarrollados para un sistema particular y las lecciones extraidas a través de su construcción y uso son frecuentemente más generales que el sistema mismo. Para comunicar y reusar tales conceptos, ayuda abstraer y especificar sus caracteristicas esenciales. La producción de software podria ser más sistemática si cada proyecto registrara las especificaciones de los conceptos clave del sistema para un uso futuro" [GUTT 82].

Nos parece que esta es una de las ventajas más importantes. La posibilidad de generalizar aspectos especificados formalmente facilita un desarrollo más efectivo de investigaciones posteriores al proporcionar una base teórica confiable.

III.1.3 ALGUNAS OBJECIONES.

Acerca de las desventajas generales de especificar formalmente los impugnadores opinan que la principal de ellas es la dificultad de la notación matemática empleada. Subyace en esto la idea de que sólo los expertos o profesionales en matemáticas pueden entender y escribir especificaciones formales, lo cual consideramos que es falso. Se requiere si un entrenamiento previo en el leguaje a

utilizar que puede ser de muy corta duración si el interesado tiene bases firmes de matemática elemental (lógica y conjuntos) y si sólo se trata de entender las especificaciones. Cuando el propósito es lograr la capacidad de producir especificaciones formales, el proceso de aprendizaje puede ser un poco más largo.

Escribir especificaciones formales no es fácil, pero tampoco es demasiado difícil. Hay que tomar en cuenta que en el proceso de especificar formalmente lo más difícil es haber entendido muy bién lo que se desa especificar.

Otra objeción aducida por profesionales en la programación de sistemas es la frecuente necesidad de cambios en los sistemas por factores internos y externos. Sobre esto argumentamos que es mucho más productivo y efectivo hacer cambios cuando se tiene la especificación formal, pues cuando el factor de cambio es interno - como por ejemplo cuando se encuentra que la arquitectura es difícil de implementar - la especificación es una base muy firme para el cambio y ahorra trabajo; cuando el cambio es obligado por factores externos que obligan a una tranformación total del sistema puede resultar obsoleta la especificación formal hecha, pero sin embargo los conceptos registrados pueden generalizarse y ser de mucha utilidad.

La última objeción que discutiremos es "que el usuario no entiende una especificación formal". Aunque esto puede ser muy cierto, siempre es posible elaborar manuales o docu-

mentos mucho más claros, explícitos y entendibles, a partir de una especificación formal. Sucede frecuentemente que en dichos documentos informales se hereda gran parte de la estructura del modelo formal. Por ejemplo, la descripción informal del modelo orientado a objetos del capítulo anterior solo pudo desarrollarse tomando como base la especificación formal. Por otro lado, es un hecho que en la actualidad ya existen usuarios interesados en comprender directamente las especificaciones formales y la tendencia es que en un futuro cercano esta necesidad será generalmente admitida.

El rechazo no razonado al empleo de las técnicas de especificación formal puede significar el abandono de una alternativa interesante ,y en pleno desarrollo, para el establecimiento de bases firmes en el desarrollo de sistemas de software.

III.2 METODOS DE ESPECIFICACION FORMAL.

Se ha establecido que un lenguaje formal es cualquier lenguaje cuya semántica sea precisa, es decir, en el cual cualquiera de sus expresiones tenga solamente un significado. También mencionamos que un lenguaje de especificación formal es un lenguaje formal diseñado expresamente para propósitos de especificación.

Puesto que existen varios aspectos que pueden ser especificados en un sistema (funcionamiento, estructura,

resultados, operación, etc.) y puesto que también existen varios propósitos y enfoques al especificar, se han diseñado varios lenguajes de especificación formal idóneos para algunos aspectos y propósitos en particular.

Existen en la actualidad varios métodos de especificación formal que varían según los propósito para los que fueron diseñados o según el lenguaje definido para ellos, aunque todos estos lenguajes tienen en común el fundamentarse en conceptos matemáticos.

En este trabajo no abordaremos una revisión de los diferentes métodos y lenguajes de especificación pues implicaría desviarnos de nuestros objetivos (36), sin embargo sí señalaremos los requisitos generales que todo método de especificación formal debe satisfacer, según Liskov [LISK 75]:

- Formalidad. Esta cualidad es aportada por el lenguaje de especificación formal que el método utilice. La formalidad implica la interpretación inambigua de las expresiones del lenguaje.
  Puesto que el fundamento de todo lenguaje de especificación formal es matemático, será posible: realizar pruebas de corrección, responder problemas matemáticos relacionados (como saber cuando hay equivalencias entre especificaciones) y el procesamiento automático de las especificaciones.
- Constructividad. Es la cualidad que permite la construcción -sin exceso de dificultad- de una especificación de los conceptos que se desee describir (suponien-

do que se conoce bién el método y que se entiende el concepto a ser especificado).

- Comprensividad. Una especificación correcta debe poder ser leída por cualquier persona entrenada en el lenguaje usado. A partir de la lectura debe ser posible reconstruir el concepto que la especificación intenta describir.
- Minimalidad. Debe ser posible, usando el método de especificación, construir especificaciones mínimas, es decir, aquellas que describan las propiedades relevantes de los conceptos y nada más.
- Amplio rango de aplicación. Esta característica es un criterio de efectividad del método. Entre más amplio es el rango de aplicación del método, éste será más útil; aunque por lo general los métodos de especificación están enfocados a ciertas áreas, hay conceptos que algunos métodos describen en forma más natural y directa.
- Extensividad. Es la propiedad de las especificaciones realizadas con algún método que facilita la modificación de la especificación de un concepto cuando éste sufre un cambio mínimo. Esta cualidad, así como las dos anteriores, es deseable aunque no indispensable.

Dedicaremos lo que resta de este capítulo a la descripción del lenguaje de especificación que hemos escogido para la formalización del modelo orientado a objetos.

III.3 INTRODUCCION AL LENGUAJE DE ESPECIFICACION Z.

En la actualidad, uno de los lenguajes de especificación formal que está siendo estudiado es un lenguaje llamado "Z". Esta notación fue creada por el investigador Jean R. Abrial de la Universidad de Oxford en el año 1980. A partir de esa fecha, el Grupo de Investigación en Programación de dicha universidad ha aplicado Z en sus proyectos de especificación, al mismo tiempo que la notación se sigue desarrollando y perfeccionando cada vez más.

Las aplicaciones realizadas utilizando Z han mostrado que el lenguaje resulta práctico (incluso en aplicaciones industriales) para un gran rango de problemas, incluyendo sistemas de información de gran tamaño y aplicaciones variadas en ingeniería de software.

El método de especificación formal que se ha desarrollado utilizando Z es presentado en forma práctica en un ejemplo más adelante.

Las principales características del método son:

- Uso del lenguaje de especificación formal Z.
- Uso de tipos de datos y operadores para la descripción de los atributos observables del sistema a especificar. Los tipos de datos y operadores deben facilitar la comprensión de la estructura y de lo que hace el sistema (no cómo lo hace), por lo cual no necesariamente deben corresponder a los de la implementación.

- Definición de predicados que relacionen variables, tipos de datos y operaciones del sistema. Estos predicados describen el comportamiento que los elementos del sistema deben mantener siempre, por lo cual se les llama invariantes (un invariante es un predicado que siempre es verdadero).
- Modelación de sucesos que ocurren en el sistema definiendo predicados que establecen el estado de los elementos del sistema antes del suceso (a estos predicados se les llama precondiciones) y después de él (a estos predicados se les llama postcondiciones).
- Inclusión de comentarios complementarios en lenguaje natural que establezcan precisiones de contexto e interpretación para facilitar la comprensión de la especificación. El lenguaje natural empleado en estos comentarios debe ser preciso, es decir, a cada concepto y designación introducidos debe dársele siempre el mismo significado.

Puesto que este método cumple con los requisitos generales que debe satisfacer un método de especificación formal efectivo, además de que se adapta bién para sistemas grandes, como es el caso de bases de datos, es el que utilizaremos para la especificación formal del modelo orientado a objetos.

El lenguaje Z consiste esencialmente de una notación

matemáticas junto con una notación de esquemas. Los esquemas permiten presentar el texto matemático de la especificación formal en una forma organizada, estructurada y práctica.

En las secciones siguientes se presenta una versión reducida de la notación Z incluyendo todos los recursos que vamos a utilizar. El interesado en profundizar en el conocimiento de Z podrá consultar próximamente [HAYE 86], aunque por el momento no hay ninguna publicación accesible en el mercado; las dos últimas secciones de este capítulo han sido extraídas de la referencia de Z citada.

III.3.1 NOTACION MATEMATICA.

DEFINICIONES Y DECLARACIONES.

Sean x_1, x_k identificadores y T_1, T_k conjuntos.

LHX ≡ RHS	Definición de LHS como sintácticamente equivalente a RHS.
x: T	Declaración de x como tipo T.
[A,B]	Introducción a conjuntos genéricos.

LOGICA.

Sean P, Q predicados y D declaraciones.

VERDADERO, FALSO	Constantes lógicas.
~ P	Negación.
P & Q	Conjunción " P y Q ".
P v Q	Disyunción " P o Q ".
P ==> Q	Implicación: "P implica Q"
P <==> Q	Equivalencia: "P es lógicamente equivalente a Q".
∀ x: T . P	Cuantificación universal: "para toda x de tipo T, P se cumple".
∃ x: T . P	Cuantificación existencial: "existe una x de tipo T tal que P".

$\forall x_1 : T_1 ; x_2 : T_2 ; \ldots x_n : T_n \ . \ P$

"Para toda x_1 de tipo T_1, x_2 de tipo T_2 ,... , y x_n de tipo T_n, P se cumple"

$t_1 = t_2$ Igualdad entre términos.

$t_1 \neq t_2$ $\equiv \neg(t_1 = t_2)$

CONJUNTOS.

Sean S, T y X conjuntos; t, t_k términos; P un predicado y D declaraciones.

$t \in S$ Membrecía de un conjunto: "t es un elemento de S".

$t \notin S$ $\equiv \neg(t \in S)$

$S \subset T$ Inclusión de conjuntos:

$\equiv (\forall x: S \ . \ x \in T)$

$\langle\rangle$ El conjunto vacio.

$\langle t_1, t_2, \ldots, t_n \rangle$

El conjunto que contiene a $t_1, t_2, \ldots, t_n$.

$\langle x: T / P \rangle$ El conjunto que contiene exactamente a aquellas x de tipo T para las cuales P se cumple.

$(t_1, t_2, \ldots, t_n)$

Enesda ordenada de t_1 a t_n.

$T_1 \times T_2 \times \ldots \times T_n$

Producto cartesiano: el conjunto de todas las enedas tales que su K-ésima componente es de tipo T_k.

$\mathbb{P} S$ El conjunto potencia: el conjunto de todos

los subconjuntos de S.

F S — El conjunto de subconjuntos finitos de S:
≡ { T: P S / T es finito }

S ∩ T — Intersección de conjuntos: dados S, T: P X,
≡ { x: X / x ∈ S ∧ x ∈ T }

S ∪ T — Unión de conjuntos: dados S, T: P X,
≡ { x: X / x ∈ S ∨ x ∈ T }

S − T — Diferencia de conjuntos: dados S, T: P X,
≡ { x: X / x ∈ S ∧ x ∉ T }

∪ SS — Unión de conjuntos distribuida: dado
SS: P (P X) ,
≡ { x: X / (∃S: SS . x ∈ S) }

#S — Número de elementos distintos de un conjunto finito.

NUMEROS

N — El conjunto de números naturales (enteros no negativos).

N^{+} — El conjunto de números naturales estrictamente positivos.

Z — El conjunto de enteros (positivos, cero y negativos).

m..n — El conjunto de enteros entre m y n inclusive:
≡ { k: Z / m ≤ k ∧ k ≤ n }.

RELACIONES

Una relación es modelada por un conjunto de pares ordenados, por lo tanto, los operadores definidos para conjuntos pueden ser usados sobre relaciones .

Sean X, Y y Z conjuntos; x: X; y: Y; y R: X <---> Y.

X <---> Y	El conjunto de relaciones de X a Y: ≡ P (X × Y).
x R y	x está relacionada con y por R: ≡ (x,y) ∈ R.
x |--> y	≡ (x,y)
$\{ x_1 \mapsto y_1 , x_1 \mapsto y_2 , \ldots , x_2 \mapsto y_n \}$	La relación $\{ (x_1 ,y_1), \ldots , (x_n ,y_n) \}$.
dom R	El dominio de la relación: ≡ {x: X / (∃ y: Y . x R y)}.
rng R	El rango de la relación: ≡ {y: Y / (∃ x: X . x R y)}.
$R_1 ; R_2$	Composición relacional hacia atrás: Dados $R_1 : X \leftrightarrow Y$, $R_2 : Y \leftrightarrow Z$, ≡ $\{x:X, y:Y / (\exists y: Y . x R_1 y \ \& \ y R_2 z)\}$
R^{-1}	Inversa de la relación R ≡ { y: Y, x: X / y R x }
id X	Función identidad sobre el conjunto X ≡ { x: X . x |--> x }.
R^k	La relación R compuesta con ella misma k veces: dada R: X <---> X, $R^0 \equiv$ id X, $R^{k+1} \equiv R ; R^k$.
R^*	Cerradura transitiva reflexiva: ≡ $\bigcup \{ n: N . R^n \}$
R^+	Cerradura transitiva no reflexiva: ≡ $\bigcup \{ n: N^+ . R^n \}$.
S ◁ R	Restricción de dominio a S: Dado S: P X,

$\equiv \{ x: X, y: Y / x \in S \ \& \ x \ R \ y \}$.

$S \lhd R$ Sustracción de dominio: Dado $S: P X$,

$\equiv (X - S) \lhd R$.

$R_1 \ ⊕ \ R_2$ Sobre-escritura: dados $R_1, R_2 : X \leftrightarrow Y$,

$\equiv (\text{dom } R_2 \lhd R_1) \cup R_2$

FUNCIONES.

Una función es una relación con la propiedad de que para cada elemento en su dominio hay un único elemento en su rango relacionado con él. Como las funciones son relaciones, todos los operadores definidos antes para relaciones también se aplican para funciones.

$X \dashrightarrow Y$ El conjunto de funciones parciales de X a Y

$\equiv \{f: X \rightarrow Y / \forall x: \text{dom } f \cdot (\exists! y:Y \cdot x \ f \ y)\}$.

$X \rightarrow Y$ El conjunto de funciones totales de X a Y:

$\equiv \{ f: X \dashrightarrow Y / \text{dom } f = X \}$.

$X \rightarrowtail Y$ El conjunto de funciones parciales uno a uno de X a Y:

$\equiv \{ f: X \rightarrowtail / \forall y: \text{rng } f \cdot (\exists! x:X \cdot x \ f \ y)\}$

$X \rightarrowtail Y$ El conjunto de funciones totales uno a uno de X a Y:

$\equiv \{ f: X \rightarrowtail / \text{dom } f = X \}$.

$X \twoheadrightarrow Y$ El conjunto de funciones parciales finitas de X a Y.

$\equiv \{ f: X \twoheadrightarrow / f \in F (X \times Y) \}$.

$f \ t$ La función f aplicada a t.

SECUENCIAS.

seq X	El conjunto de secuencias cuyos elementos son obtenidos de X: $\equiv \{ A: \mathbb{N}^{+} \rightarrow X \mid \mathrm{dom}\ A = 1..\#A \}$.
#A	La longitud de la secuencia A.
[]	La secuencia vacia <>.
$[a_1, \ldots, a_n]$	$\equiv \{ 1 \mapsto a_1, \ldots, n \mapsto a_n \}$.
$[a_1, \ldots, a_n]\ \hat{}\ [b_1, \ldots, b_n]$	Concatenación: $\equiv [a_1, \ldots, a_n, b_1, \ldots, b_n]$
head A	$\equiv A(1)$.
last A	$\equiv A(\#A)$.

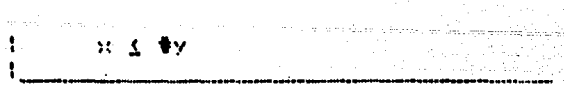
III.3.2 NOTACION DE ESQUEMAS.

Definición de un esquema: Un esquema agrupa algunas declaraciones de variables y predicados que relacionan esas variables. Hay dos formas de escribir esquemas: verticalmente, por ejemplo

S

x: N

y: seq N



u horizontalmente, por ejemplo

S ≡ [x: N ; y: seq N / x ≤ ♦y].

Esquemas como tipos.

Cuando un nombre de un esquema S es usado como un tipo, esto establece que para todo el conjunto de objetos descrito por el esquema se cumplen los predicados, por ejemplo, w: S declara una variable w con componentes x (número natural), y (secuencia de números naturales) tales que x ≤ y.

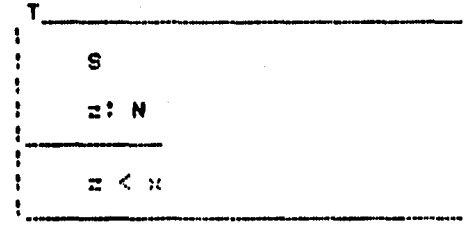
Proyecciones.

Los nombres de componentes de un esquema pueden ser usados como proyecciones, por ejemplo, dado w: S, w.x es el componente x de w, w.y es su componente y; por supuesto el siguiente predicado se cumple: w.x ≤ w.y .

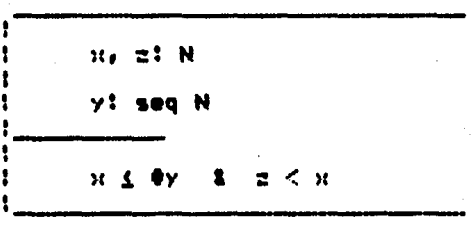
Inclusión.

Un esquema S puede ser incluído dentro de las declaraciones de un esquema T cualquiera (T puede ser igual a S). Las declaraciones de S serán añadidas a

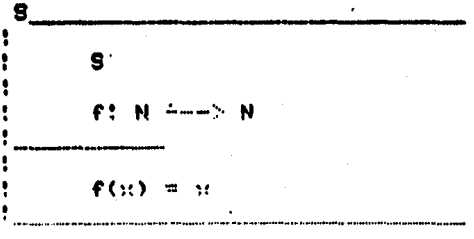
las declaraciones de T (variables declaradas en S y T deben ser del mismo tipo) y los predicados de S y T son conjuntados, por ejemplo



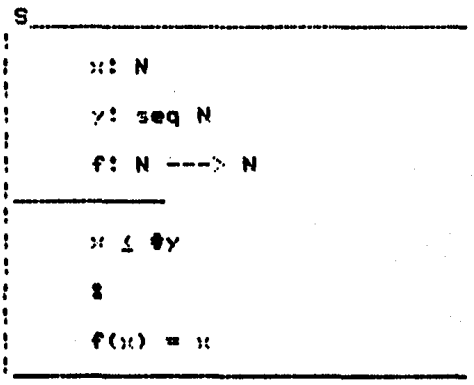
es equivalente a



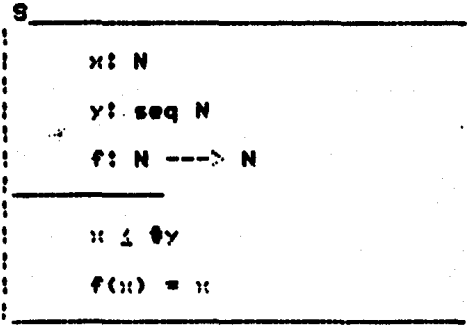
La inclusión puede usarse para redefinir, ampliando un esquema determinado, por ejemplo



es equivalente a



Cuando tengamos conjunciones de predicados como en el esquema anterior, omitiremos el símbolo de conjunción así



Renombramiento de componentes.

S[nuevo/viejo] denota al esquema S cuyo componente

llamado 'viejo' es renombrado por 'nuevo' en su declaración y en todo lugar donde aparezca 'viejo' en predicados. Por ejemplo S[x/x] es [x:N, y: seq N, f: N ---> N / x ≤ #y & f(x)=x].

Decoración.

Los esquemas decorados con alguna marca, como apóstrofo, subindice, etc. implican el renombramiento sitemático de las variables declaradas en el esquema. Por ejemplo, S' es [x': N, y': seq N, f': N --> N / x' ≤ y' & f'(x')=x'].

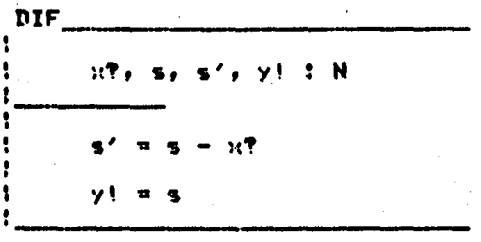
Esquemas de operaciones.

Las siguientes convenciones son usadas para nombres de variables en aquellos esquemas que representan operaciones:

sin apóstrofo	estado anterior a la operación,
con apóstrofo	estado posterior a la operación,
terminación en "?"	parámetro de entrada de la operación.
terminación en "!"	parámetro de salida de la operación.

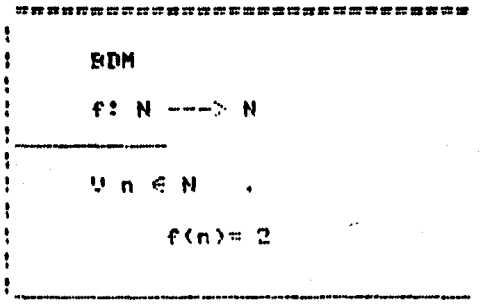
Un ejemplo se muestra en el siguiente esquema donde el parámetro de entrada es x?, el de salida es y!, s es el estado de una variable antes de la operación DIF y s' es el estado de la variable después

de la operación.



Selectores.

Un selector es una función auxiliar que es redundante en la especificación pero que permite facilitar la referencia a ciertos componentes de un esquema. Suponiendo que el esquema al que se hace referencia se llama BDM, un selector f se especificará incluyendo a BDM, declarando la signatura de f y en la parte de predicados incluyendo la definición de f, como se muestra en el ejemplo



CAPITULO IV.

FORMALIZACION DEL MODELO BASICO.

En este capitulo y los dos siguientes se presenta la especificación formal del modelo de datos orientado a objetos que se ha descrito ya, informalmente, en el capitulo II.

En adelante, a una base de datos cuyo modelo conceptual sea el orientado a objetos, le llamaremos Base de Datos Molecular (BDM). En la especificación que se expondrá se formalizan los conceptos básicos de BDM, distinguiendo estrictamente sus dos partes: esquema e instancia, a los cuales llamaremos respectivamente EBDM e IBDM.

El trabajo ha sido desarrollado utilizando el lenguaje de especificación formal llamado 'Z'. Por lo general, en la presentación de los conceptos éstos se introducen con algunos comentarios, y en seguida se formalizan en 'Z'. Algunas veces, la formalización se acompaña de lo que hemos llamado Nota Técnica, en la cual se hacen observaciones relacionadas con el lenguaje usado y sus implicaciones (aquellos entrenados en la notación matemática pueden omitir la lectura de dichas notas). Finalmente, cuando se considera necesario, se ejemplifican los conceptos expuestos por lo cual se recomienda al lector, que cuando no comprenda muy bien la especificación en 'Z', consulte el ejemplo y reconsidere la especificación.

IV.1 DEFINICION DEL ESQUEMA DE BDM.

Un Esquema de Base de Datos Molecular (EBDM) define las estructuras y propiedades de las clases de objetos del modelo de datos. A continuación presentamos una secuencia de esquemas con descripciones sucesivamente más precisas del EBDM.

En primer lugar se exponen algunas consideraciones sobre los valores de la BDM y sus dominios, en seguida se define una clase de objetos y sus atributos primitivos con lo que se puede dar la primera aproximación a EBDM. Por último se definen en forma general los conceptos de agregados y roles.

IV.1.1 VALORES Y DOMINIOS.

Empezamos nuestra descripción introduciendo el conjunto de todos los posibles valores que pueden estar almacenados en la base de datos al cual denominaremos como VALOR. En este nivel de abstracción no es relevante detallar la forma en que los valores serán representados.

[VALOR];

Nota Técnica.- Se introduce el conjunto VALOR. La especificación queda parametrizada por las diversas representaciones que se le asignen a este conjunto; de esta forma la especificación es más general.

Cada valor almacenado en la base de datos debe pertenecer a algun dominio definido. Llamaremos NOM-DOM al conjunto de todos los posibles nombres de dominios. La asignación de conjuntos de valores a nombres de dominios será modelada por la función Val-Dom.

[NOM-DOM];

Val-Dom: NOM-DOM --//--> P VALOR

Nota Técnica.- Función parcial porque no agota todos los posibles nombres de dominios. Finita ya que los dominios que se definan serán finitos. El conjunto de valores asociados a un dominio puede ser infinito (por ejemplo: ENTEROS !--> {0,1,..?}). Un valor puede pertenecer a más de un dominio.

IV.1.2 CLASES Y ATRIBUTOS PRIMITIVOS.

Toda la información en BDM se encuentra organizada en objetos, cada objeto pertenece a una clase de objeto. La definición de cada clase determina y describe la estructura de los objetos que pertenecerán a dicha clase. Para modelar esto, introduciremos el primer esquema, que llamaremos DEF-CLASE, en donde se presentan los componentes más elementales de una definición de clase y sus predicados o invariantes asociados. Más adelante introduciremos otros componentes que corresponderán a conceptos más elaborados a través de subsecuentes refinamientos de DEF-CLASE.

La caracterización básica de un objeto se establece a través de los atributos que lo califican. Denominamos atributos primitivos a aquellos atributos del objeto que no dependen de ningún parámetro, es decir, que no están en función ni de atributos de otros objetos, ni de atributos propios.

Los atributos primitivos de una clase quedarán definidos por una función, llamada AP, que asocia a cada nombre de atributo el nombre del dominio al que pertenece su valor.

Puesto que en ocasiones será necesario hacer referencia al conjunto de nombres de atributos de una clase, que llamaremos Atr-Prim, definiremos esta variable en el esquema, aunque estrictamente no es necesaria ya que es redundante (38). En varias ocasiones en adelante se introducen en los esquemas variables que son redundantes, a este tipo de variables les llamaremos derivables.

A continuación se introduce el conjunto de todos los posibles nombres de atributos, al cual denominaremos NOM-ATR, y en seguida se presenta el esquema de DEF-CLASE.

[NOM-ATR];

DEF-CLASE

AP: NOM-ATR --//--> NOM-DOM

Atr-Prim: F NOM-ATR

---

rng AP C dom Val-Dom I1

dom AP = Atr-Prim I2

I1: Los nombres de dominios deben tener asociados conjuntos de valores.

I2: Definición del conjunto de nombres de atributos primitivos. Todos los atributos primitivos deben tener un dominio asociado.

Nota Técnica.- AP es función parcial ya que no agota todos los posibles nombres de atributos, es finita ya que los nombres de atributos de una clase serán finitos (39). Atr-Prim es derivable.

Ejemplo.

Supongamos una clase donde se defina el siguiente componente

AP = { RFC !--> CUERDA, SUELDO !--> ENTERO }

donde RFC y SUELDO pertenecen a NOM-ATR, y donde CUERDA y ENTERO pertenecen a NOM-DOM y además donde esté definido

Val-Dom = { CARACTER !--> {'a',...,'z','-','/','1',...},
CUERDA !--> Seq CARACTER,
ENTERO !--> { 1,2,...} }

Entonces

rng AP = { CUERDA, ENTERO } C dom Val-Dom
dom AP = { RFC, SUELDO } = Atr-Prim

y por lo tanto se satisfacen los invariantes I1, I2.

Cada definición de clase debe corresponder a un nombre de clase, perteneciente al conjunto de todos los po-

sibles nombres de clases, al cual llamaremos NOM-CLASE

La primera aproximación a la definición de EBDM debe establecer una asociación entre los nombres de las clases y sus definiciones, a esta función le llamaremos Esq.

[NOM-CLASE];

EBDM

Esq: NOM-CLASE --//--> DEF-CLASE

Nota Técnica.- A cada nombre de clase se le asocia una y sólo una definición de clase, puesto que Esq es función. Puede haber dos nombres de clases que tengan asociada la misma definición, ya que la función no es inyectiva.

Ejemplo.

Un EBDM podría estar definido por la siguiente función Esq

dom Esq = { EMPLEADO }

Suponemos que EMPLEADO es un nombre de clase y que Esq(EMPLEADO) tiene los componentes

Esq(EMPLEADO).AP = {RFC !--> CUERDA,SUELDO !--> ENTERO}

Esq(EMPLEADO).Atr-Prim = {RFC, SUELDO}

IV.1.3 AGREGADOS Y ROLES.

Una BDM admite objetos complejos, es decir, objetos que pueden estar asociados a conjuntos de objetos. La

única forma de establecer asociaciones entre objetos en RDM es a través de roles.

En la definición de una clase, la función que llamaremos Agreg-Rol modelará la asociación de la clase definida con otras clases, a través de roles. Así, Agreg-Rol asociará a cada nombre de rol un conjunto finito de nombres de clases, al cual llamaremos Agregados. Al conjunto de nombres de roles que tienen asociados agregados le llamaremos Roles.

Para modelar los nuevos conceptos, redefiniremos el esquema DEF-CLASE, incluyendo antes el conjunto de todos los posibles nombres de roles, al cual denominaremos NOM-ROL.

[NOM-ROL];

DEF-CLASE

DEF-CLASE	
Agreg-Rol: NOM-ROL --//--> F NOM-CLASE	
Roles: F NOM-ROL	
Agregados: F NOM-CLASE	
Roles = dom Agreg-Rol	I3
Agregados = U rng Agreg-Rol	I4

I3: Definición del conjunto de nombres de roles de la clase. Todos los roles deben tener un conjunto de clases asociadas.

I4: Definición del conjunto de agregados de una

clase como la unión distribuida del rango de Agreg-Rol.

Nota Técnica.- La inclusión de DEF-CLASE en el esquema significa la redefinición de DEF-CLASE incluyendo las variables y predicados que tenía en el esquema anterior (40). Los componentes: Roles y Agregados son derivables. El conjunto Roles podría ser vacío cuando el dominio de Agreg-Rol lo es, en este caso la clase tampoco tendría agregados. Los conjuntos del rango de Agreg-Rol no son necesariamente disjuntos pues no se establece ninguna limitación al respecto. Es posible que la clase definida pertenezca a su conjunto Agregados (recursividad).

Ejemplo.

La definición de una clase que llamaremos c4 podría estar dada por los siguientes componentes

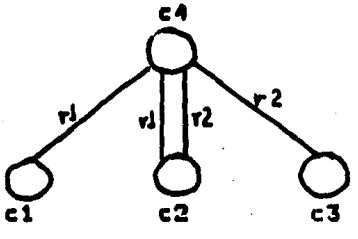
$$Esq(c4).Agreg\text{-}Rol = \{ r1 \mapsto \{c1,c2\} , \; r2 \mapsto \{c2,c3\} \}$$

$$Esq(c4).Roles = \{ r1, r2 \}$$

$$Esq(c4).Agregados = \{ c1, c2, c3 \}$$

donde r1 y r2 pertenecen a NOM-ROL; c1, c2, c3 y c4 pertenecen a NOM-CLASE.

La gráfica que ilustra a c4 con sus agregados es



Observemos que c4 no tiene definidos atributos primitivos, además, note que dos roles (o más) pueden asociar a un mismo agregado, como en este caso r1 y r2.

En el ejemplo anterior podemos observar que al definir los agregados de una clase, estamos haciendo referencia a otros nombres de clases. Es necesario que esos nombres tengan ya asociada una definición de clase, es decir, deben pertenecer al dominio de Esq.

Para modelar esto, es necesario redefinir EBDM especificando que para toda clase definida se cumpla que sus agregados estén ya definidos.

EBDM

EBDM	
∀ c: dom Esq .	
Esq(c).Agregados ⊂ dom Esq	I5

I5: Todos los agregados deben ser clases definidas.

IV.2 DEFINICION DE INSTANCIA DE BDM.

Una Instancia de una Base de Datos Molecular (IBDM) es un conjunto de valores determinado, que corresponde a a las clases de objetos definidas en EBDM.

Sabemos que una IBDM que corresponda a un EBDM particular definido, constituye lo que hemos llamado una BDM. Veremos que el problema fundamental en el estudio de las características de una IBDM que realmente corresponda a

una EBDM, será determinar cuales son los invariantes necesarios para asegurar la consistencia entre IBDM y EBDM.

En seguida se mostrará una primera aproximación a la definición de una instancia de clase y a la definición de IBDM, especificando los primeros invariantes necesarios para la consistencia; en esta primera aproximación se asumirá que el conjunto de Roles es vacio para toda clase definida en el EBDM.

A continuación veremos qué se entiende por instanciar un rol y se refinarán más los esquemas incluyendo este concepto.

IV.2.1 INSTANCIAS DE CLASES.

Cada clase definida en EBDM podrá tener asociado un conjunto de instancias u objetos de esa clase, llamaremos INST-CLASE al esquema que definirá dicho conjunto de objetos. Cada instancia será identificada de manera única por medio de un identificador interno (asignado por el sistema) al cual el usuario no tiene acceso. Al conjunto de todos los posible identificadores de instancia le llamaremos ID-INST.

El esquema INST-CLASE estará formado por las siguientes variables:

- Una función, que denominaremos IC, que asociará a cada identificador de instancia los atributos primitivos instanciados asociados a sus respectivos valores.
- Un componente derivado, que denominaremos Instancias,

que será el conjunto de identificadores de instancias de una clase.

En seguida se formaliza el esquema INST-CLASE, introduciendo previamente el conjunto ID-INST.

[ID-INST];

INST-CLASE

IC: ID-INST --//--> (NOM-ATR --//--> VALOR)

Instancias: F ID-INST

dom IC = Instancias I6

I6: Definición del conjunto de identificadores de instancias de una clase.

Nota Técnica.- El rango de la función IC es a su vez otra función. La variable Instancias es derivable.

Ejemplo.

Supongamos que se tiene la siguiente INST-CLASE, que corresponderá, como veremos más adelante, a la clase EMPLEADO con los siguientes componentes

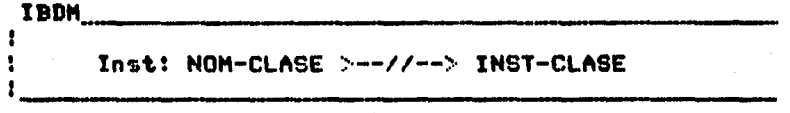
IC = ⟨id1 !--> ⟨RFC !--> GUNP500622,SUELDO !--> 100⟩,
id2 !--> ⟨RFC !--> SARM600308,SUELDO !--> 200⟩ ⟩

Instancias = ⟨ id1, id2 ⟩

donde id1, id2 pertenecen a ID-INST; RFC y SUELDO pertenecen a NOM-ATR y donde suponemos que

VALOR ≡ Seq CARACTER

En el ejemplo anterior expusimos una instancia que corresponde a la clase EMPLEADO, para especificar formalmente esta correspondencia en general para todas las clases que tengan instancias, definiremos un esquema, al que llamaremos IBDM, que será una primera aproximación a la definición de una instancia de base de datos molecular. IBDM tendrá como comoponente a la función Inst, la cual asociará a nombres de clase, instancias de ella.



Nota Técnica.- Puesto que Inst es inyectiva no pueden tenerse dos nombres de clase asociados a la misma instancia. Aunque se diera el caso de que dos nombres de clase se asociaran a la misma definición sus INST-CLASE diferirían por lo menos en sus identificadores de instancia.

Hemos comentado que los identificadores de instancia son únicos en toda la base de datos, sin embardo, este hecho todavía no se encuentra reflejado en el modelo. El siguiente refinamiento de IBDM establece que para cualquier par de nombres de clase diferentes en el dominio de Inst, sus respectivos conjuntos de identificadores de instancia deben ser disjuntos.

IBDM

IBDM

$$\forall c_i, c_j : \text{dom Inst} \cdot (c_i \neq c_j) \Rightarrow \text{Inst}(c_i).\text{Instancias} \cap \text{Inst}(c_j).\text{Instancias} = \{\} \quad \text{I7}$$

I7: Todos los identificadores de instancias son diferentes.

Nota Técnica.- Cuando la intersección de dos conjuntos es vacía se dice que éstos son disjuntos.

Ejemplo.

Si llamamos INST-CLASE-EMPLEADO al esquema INST-CLASE del ejemplo anterior, entonces IBDM podría estar definido por la siguiente función

Inst = {EMPLEADO |--> INST-CLASE-EMPLEADO,
CLASE-X |--> INST-CLASE-X}

donde EMPLEADO y CLASE-X pertenecen a NOM-CLASE y donde INST-CLASE-X podría estar dado por

Inst(CLASE-X).IC = {id3 |--> {ATR1 |--> val1},
id4 |--> {ATR1 |--> val2} }

Inst(CLASE-X).Instancias = {id3, id4}

donde id3, id4 son ID-INST; ATR1 es un NOM-ATR val1,val2 suponemos que pertenecen a VALOR.

Observe que

Inst(EMPLEADO).Instancias = {id1, id2}

∩

Inst(CLASE-X).Instancias = {id3, id4} = {}

IV.2.2 CONSISTENCIA EN BDM

Para tener una BDM consistente es necesario que se establezca una correspondencia determinada entre su EBDM y su IBDM.

Analizando los ejemplos presentados podemos inducir que un primer requisito de consistencia sería establecer que cada nombre de clase que tenga asociada una instancia debe estar previamente definida en EBDM, ya que en caso contrario no se podría determinar la consistencia de las instancias con sus clases .

En el siguiente esquema introducimos el concepto de BDM formalmente, y definimos como invariante el requisito mencionado en el párrafo anterior. BDM debe proporcionar un marco que incluya tanto a EBDM como a IBDM.

BDM
EBDM
IBDM
dom Inst C dom Esq — I8

I8: Los nombres de las clases instanciadas son un subconjunto de los nombres de clases definidos.

Nota Técnica.- Este esquema, al incluir en sus decla-

raciones las últimas versiones de EBDM e IBDM, hace referencia indirecta a todo lo especificado hasta el momento.

Ejemplo.

Suponiendo que IBDM está definida como en el ejemplo anterior, para que se satisfaga I8 debe cumplirse que

dom Inst = ⟨EMPLEADO, CLASE-X⟩ ⊂ dom Esq

Para poder especificar más claramente todos los demás requisitos de consistencia necesarios entre una DEF-CLASE y su INST-CLASE correspondiente, definiremos a continuación un conjunto de selectores auxiliares.

Recordemos que un selector es una función redundante en la especificación pero que facilita la referencia a ciertas variables y componentes de un modelo. Los siguientes son los selectores que introduciremos y su descripción:

- Atr-Inst, dada una clase y un identificador de instancia de esa clase nos proporciona el conjunto de atributos de la instancia que tienen asociado un valor.
- Valor-Inst, dada una clase, un identificador de instancia de ella y un atributo, nos proporciona el valor del atributo en la instancia.
- Dominio, dada una clase y un atributo de ella nos proporciona el dominio definido para ese atributo.

La especificación de los selectores es la siguiente

```
BDM

Atr-Inst: NOM-CLASE X ID-INST --/--> F NOM-ATR

Valor-Inst: NOM-CLASE X ID-INST X NOM-ATR --/--> VALOR

Dominio: NOM-CLASE X NOM-ATR --/--> NOM-DOM
```

```
Atr-Inst = { (c,id) !---> dom Inst(c).IC(id) /
             c:dom Inst, id:dom Inst(c).IC }

Valor-Inst = { (c,id,a) !---> Inst(c).IC(id)(a) /
               c:dom Inst, id:dom Inst(c).IC,
               a:dom Inst(c).IC(id) }

Dominio = { (c,a) !---> Esq(c).AP(a)  /
            c: dom Esq, a: dom Esq(c).AP }
```

Nota Técnica.- Esquema que define selectores. Se icluye BDM pues es el contexto necesario para la definición. Se declaran lls selectores con su signatura y como predicados se definen los conjuntos de pares ordenados que corresponden a cada selector. En adelante todos los esquemas de selectores tendrán este formato.

Ejemplo.

Sea c igual a EMPLEADO , y dado el identificador id2 tenemos que

Atr-Inst(c. id) = Atr-Inst(EMPLEADO, id2)

= ⟨RFC, SUELDO⟩

Supongamos que nos interesa saber el valor que tiene el atributo SUELDO de la instancia id2, entonces aplicamos el selector Valor-Inst el la forma siguiente

Valor-Inst(EMPLEADO, id2, SUELDO) =

Inst(EMPLEADO).IC(id2)(SUELDO) = 200

Un ejemplo de la aplicación del selector Dominio es

Dominio(EMPLEADO, SUELDO) = Esq(EMPLEADO).AP(SUELDO)
= ENTERO

Ahora, ya tenemos herramientas para abordar la definición de dos invariantes de consistencia muy importantes.

El primero de ellos debe garantizar que todos los atributos de una instancia hayan sido declarados como atributos primitivos de la clase correspondiente. Debe notarse que no es necesario pedir que el conjunto de atributos instanciados sea igual al conjunto de atributos declarados, ya que podría darse el caso en que el conjunto de atributos instanciados sea un subconjunto propio de los atributos declarados. En este modelo no se establece ninguna limitación respecto a algún subconjunto mínimo de atributos instanciados puesto que estamos identificando de manera única a cada instancia por el identificador generado por el sistema (42).

El segundo invariante deberá garantizar que todos los valores de atributos instanciados pertenecen al conjunto de valores asociado al dominio declarado para cada atributo en la definición de la clase correspondiente.

A continuación se redefine BDM incluyendo los invariantes anteriores como predicados.

BDM

BDM

∀ (c,id) : dom Atr-Inst .

Atr-Inst(c,id) ⊂ Esq(c).Atr-Prim I9

∀ (c,id,a) : Valor-Inst .

Valor-Inst(c,id,a) ∈ Val-Dom (Dominio (c,a)) I10

I9: Consistencia en los nombres de los atributos de las instancias (Los nombres de atributos de una instancia deben ser algunos de los definidos en su clase correspondiente).

I10: Consistencia en los valores asociados a los atributos de las instancias.

Nota Técnica.- No es necesario pedir que c pertenezca al dominio de Inst puesto que estamos aplicando selectores, al indicar que el selector se está aplicando sobre toda enesda en su dominio quedan automaticamente establecidos todos los requisitos que pide el selector.

Ejemplo.

Supongamos que tenemos las instancias id5, id6 de EMPLEADO siguientes

Inst(EMPLEADO).IC(id6) = ⟨RFC :--> TOTM400730,
EDO-CIVIL :--> CASADO⟩

entonces, id6 no es una instancia válida ya que no se cumple el invariante I9 ya que

Atr-Inst(EMPLEADO, id6) = dom Inst(EMPLEADO).IC(id6)
=⟨RFC, EDO-CIVIL⟩

no esta contenido en

Esq(EMPLEADO).Atr-Prim = ‹RFC, SUELDO›

Si por otra parte suponemos también que la instancia id5 está definida por

Inst(EMPLEADO).IC(id5) = ‹SUELDO !--> 101.50›

entonces puede probarse que el invariante I9 si es válido pero I10 no lo es ya que

Valor-Inst(EMPLEADO, id5, SUELDO) =
Inst(EMPLEADO).IC(id5)(SUELDO) =
101.50 ∉ Val-Dom (Dominio (EMPLEADO,SUELDO))=
Val-Dom (Esq(EMPLEADO).AP(SUELDO))=
Val-Dom (ENTEROS)=‹1,2,3,...›

IV.2.3 INSTANCIAS DE ROLES.

Hasta el momento, la especificación de INST-CLASE define instanciación de clases en las que se supone que su conjunto de roles es vacio. Para generalizar INST-CLASE debemos describir en qué consitirá la instanciación de un rol.

Remitiéndonos al último esquema donde se define una clase (DEF-CLASE) recordemos que la función de agregación

Agreg-Rol: NOM-ROL --//--> F NOM-CLASE

determina el conjunto de roles y de agregados de una clase. Una instanciación de un rol r será esencialmente la

asociación de dos instancias, una instancia de la clase donde se define el rol y otra instancia perteneciente a alguna de las clases de Agreg-Rol(r).

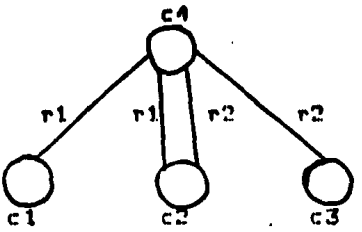
Para modelar esta situación empezaremos por redefinir a INST-CLASE incluyendo una nueva función, que llamaremos R, la cual asocia a un rol la clase que se agrega y, a su vez, la pareja de identificadores de instancias especificas asociadas.

```
INST-CLASE_____________________________________________
!
!     INST-CLASE
!
!     R: NOM-ROL --//--> NOM-CLASE --//-->
!
!                                (ID-INST <--> ID-INST)
!_____________________________________________________
```

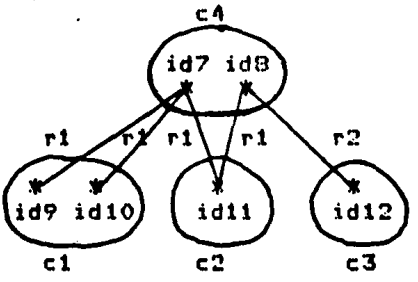
Nota Técnica.- R es una función que dado un nombre de rol, regresa otra función que al aplicarle un nombre de clase regresa una relación de identificadores asociados. Puesto que se trata de una relación, es posible tener dos parejas cuyo primer identificador sea el mismo.

Ejemplo.

Retomemos el ejemplo de la sección IV.1.3 como caso de estudio para este apartado, donde se define la clase c4 cuya gráfica es



Supongamos que la instanciación de r1 y r2 está dada en esta forma



Entonces Inst(c4), o lo que es lo mismo, el esquema INST-CLASE asociado a c4 bajo Inst, estará definido por las siguientes funciones (43).

```
Inst(c4).IC = { id7 !--> ({} !--> {}),
                id8 !--> ({} !--> {}) }                    (43)

Inst(c4).Instancias = {id7, id8}

Inst(c4).R = { r1 !--> c1 !--> (id7 !--> id9),
               r1 !--> c1 !--> (id7 !--> id10),
               r1 !--> c2 !--> (id7 !--> id11),
               r1 !--> c2 !--> (id8 !--> id11),
               r2 !--> c3 !--> (id8 !--> id12) }
```

Notemos que la última observación de la anterior nota técnica se cumple aquí ya que id7 está asociada a id9, id10 e id11, lo cual no sería posible si se hubiera modelado a los identificadores asociados mediante una función y no mediante una relación como se hizo. Note también que todos los elementos del dominio de la relación {id7, id8} son de clase c4 y los del rango {id9, id10, id11, id12} son instancias de alguno de los agregados de c4.

La función R todavía no describe completamente el concepto que deseamos modelar ya que se requiere definir algunos invariantes que garanticen la consistencia en R y que especifiquen otros requisitos necesarios como son: que los roles y clases involucradas estén bien definidos y que las instancias asociadas bajo un rol correspondan a las clases de la definición. La especificación de R será completada después de que definamos un selector auxiliar.

Para facilitar la referencia a las parejas asociadas entre dos clases definiremos un selector, al cual llamaremos Mapeo, que dada una clase c_i, un rol de ella r, y una clase c_j definida como agregado de c_i bajo r, nos dará el conjunto de parejas de instancias asociadas entre c_i y c_j. Este selector será usado más adelante, cuando se especifique el condicionamiento de Roles.

BDM

Mapeo: NOM-CLASE X NOM-ROL X NOM-CLASE -//-> (ID-INST <---> ID-INST)

$$Mapeo = \{ (c_i, r, c_j) \dashrightarrow Inst(c_i).R(r)(c_j) / c_i : dom\ Inst,\ r : dom\ Inst(c_i).R,\ c_j : dom\ rng\ Inst(c_i).R(r) \}$$

Nota Técnica.- Mapeo está definido para clases ins-

tanciadas por lo cual queda implícito que las clases han sido previamente definidas.

Ejemplo.

Aplicando Mapeo a nuestro caso de estudio (la clase c4) tenemos

Mapeo (c4, r1, c1) = {id7 !--> id9, id7 !--> id10}

Mapeo (c4, r1, c2) = {id7 !--> id11,id8 !--> id11}

Mapeo (c4, r2, c3) = {id8 !--> id12}

Todos los resultados de Mapeo son una forma simplificada de obtener Inst(c4).R .

Dos invariantes necesarios para garantizar consistencia en instanciación de roles son, en primer lugar, un predicado que establezca que los roles instanciados de una clase hayan sido previamente definidos, es decir, sean un subconjunto de los roles declarados en la definición de la clase.

En segundo lugar, se requiere un invariante que garantice que la instanciación de un rol agrega instancias de las clases definidas como agregados de ese rol.

En la siguiente redefinición de BDM se introducen los nuevos invariantes como predicados.

BDM

BDM
∀ c : dom Inst .

```
        dom Inst(c).R  C  Esq(c).Roles                    I11

    ∀ c : dom Inst  ; r : dom Inst(c).R
        dom Inst(c).R(r) ∈ Esq(c).Agreg-Rol(r)            I12
```

I11: Roles bien definidos.

I12: Las clases asociadas a través de un rol r de la clase c deben estar definidas como agregados de c a través del rol.

Ejemplo.

Comprobemos que se cumplen los invariantes anteriores para la clase c4.

dom Inst(c4).R = {r1, r2} C Esq(c4).Roles = {r1, r2}

con lo cual se cumple I11 para c = c4. Además

dom Inst(c4).R(r1) = {c1, c2} C Esq(c4).Agreg-Rol(r1) = {c1, c2}

y

dom Inst(c4).R(r2) = {c3} C Esq(c4).Agreg-Rol(r2) = {c2, c3}

con lo cual se cumple I12 para c = c4 y para todo r que pertenezca al dominio de Inst(c4).R

Antes de presentar los dos últimos invariantes que se requerirán para garantizar la consistencia en la instanciación de roles, definiremos dos selectores auxiliares que

nos facilitarán dicha tarea.

El selector Padres aplicado a cada terna (c_i, r, c_j) del dominio de Mapeo nos dará las instancias de clase c_i que son "padres" a través del rol r de instancias de clase c_j, es decir, nos dará el dominio de Mapeo (c_i, r, c_j).

El selector Hijos aplicado a la misma terna nos dará el conjunto de instancias de clase c_j que son "hijos" a través del rol r de instancias de clase c_i, es decir, nos dará el rango de Mapeo(c_i, r, c_j).

BDM

Padres: NOM-CLASE X NOM-ROL X NOM-CLASE -//-> F Id-Inst

Hijos: NOM-CLASE X NOM-ROL X NOM-CLASE -//-> F Id-Inst

$$\text{Padres} = \{ (c_i, r, c_j) \longrightarrow \text{dom Inst}(c_i).R(r)(c_j) = \text{dom Mapeo}(c_i, r, c_j) \;/\; (c_i, r, c_j) : \text{dom Mapeo} \}$$

$$\text{Hijos} = \{ (c_i, r, c_j) \longrightarrow \text{rng Inst}(c_i).R(r).(c_j) = \text{rng Mapeo}(c_i, r, c_j) \;/\; (c_i, r, c_j) : \text{dom Mapeo} \}$$

Nota Técnica.- La especificación de los conjuntos a los cuales deben pertenecer los elementos de la terna se indica en el esquema que define a Mapeo. Cuando pedimos que (c_i, r, c_j): dom Mapeo se implica que se cumplen dichas pertenencias.

Ejemplo.

Aplicaremos los selectores Padres e Hijos a nuestro caso de estudio

```
Padres(c4,r1,c1) = dom Mapeo(c4,r1,c1)
                 = dom {id7 |--> id9, id7 |--> id10}
                 = {id7}
Padres(c4,r1,c2) = dom Mapeo(c4,r1,c2)
                 = dom {id7 |--> id11, id8 |-->id11}
                 = {id7, id8}
Hijos(c4,r1,c1)  = rng Mapeo(c4,r1,c1)
                 = rng {id7 |--> id9, id7 |--> id10}
                 = {id9, id10}
Hijos(c4,r1,c2)  = rng Mapeo(c4,r1,c2)
                 = rng {id7 |--> id11, id8 |-->id11}
                 = {id11}
```

Consulte la gráfica de c4 para tener una visualización de lo que hacen estos selectores.

Los siguientes invariantes completarán los requisitos de consistencia en instanciación de roles.

El invariante I13 establecerá que todos los "padres" de una clase c_i a través del rol r de una clase c_j, son realmente instancias que pertenecen a $Inst(c_j).Instancias$. En seguida se presenta la redefinición de INST-CLASE incluyendo el invariante.

INST-CLASE

INST-CLASE

$$\forall (c_i, r, c_j) : dom\ Padres \cdot Padres(c_i, r, c_j) \subseteq Inst(c_j).Instancias \quad I13$$

I13: Las instancias que son 'padre' a través de un rol r de la clase c_i deben ser instancias de c_i.

El último invariante que presentaremos en este capítulo garantizará que todos los 'hijos' de una clase c_i a través del rol r de una clase c_j, son realmente instancias que pertenecen al conjunto definido por $Inst(c_j).Instancias$. A continuación se redefine IBDM incluyendo este invariante

IBDM

IBDM

$$\forall (c_i, r, c_j) : dom\ Hijos \cdot Hijos(c_i, r, c_j) \subseteq Inst(c_j).Instancias \quad I14$$

I14: Los 'hijos' a traves de un rol r de la clase c_i son instancias de clase c_j.

CAPITULO V.

FORMALIZACION DEL MODELO EXTENDIDO.

A partir de los conceptos fundamentales definidos en la formalización del modelo básico, en este capítulo se construirá una extensión del modelo, que incluirá los mecanismos que permitan establecer reglas de herencia, tipos de jerarquías y condicionamientos sobre agregaciones. Sabemos ya de la importancia que este tipo de capacidades tienen en la riqueza semántica de un modelo.

En la primera sección se presentará una extensión del concepto de atributos incluyendo la definición de tres nuevas categorías de ellos: atributos generados, heredados y retribuídos. En la segunda sección se incluye el concepto de roles incluyendo la capacidad para condicionarlos.

V.1 DEFINICION DE OTRAS CATEGORIAS DE ATRIBUTOS.

En la definición de una clase se ha especificado formalmente hasta el momento sólo una categoría de atributos, los atributos primitivos. Las otras categorías de atributos que formalizaremos en esta sección (generados, heredados y retribuídos) tienen como característica común el estar definidos por fórmulas que al evaluarse producen el valor correspondiente al atributo.

Dichas fórmulas deben ser expresiones bien formadas pertenecientes a un lenguaje especialmente diseñado de acuerdo a las capacidades de cómputo que se deseen. En el Apéndice II se especifica formalmente un lenguaje de expresiones con algunas capacidades fundamentales, sin embargo, para los propósitos de esta sección basta exponer informalemente los conceptos del lenguaje de expresiones que serán necesarios más adelante (44).

V.1.1 LENGUAJE DE EXPRESIONES.

El lenguaje utilizado para la definición de atributos consiste de un conjunto de expresiones bien formadas parametrizadas por el conjunto de nombres de variables que pueden aparecer en dichas expresiones. Esto implica que puede generarse una familia de lenguajes distintos dependiendo del tipo de variables que se utilicen, es decir, a cada instanciación del conjunto de nombres de variables, al cual llamaremos NOM-VAR, corresponderá un lenguaje de la familia determinado.

Al conjunto de todas las posibles expresiones bien formadas le llamaremos EXPBF y al conjunto de expresiones bien formadas parametrizado por NOM-VAR le llamaremos EXPBF[NOM-VAR].

Cada lenguaje de la familia tendrá en común los siguientes elementos:

- Mismo conjunto de operaciones. Por ejemplo suma, resta multiplicación, división, concatencación, etc.

- Dada una expresión bien formada, la función Variables aplicada a ella proporcionará el conjunto de nombres de variables de dicha expresión

  Variables: EXPBF[NOM-VAR] --/--> F NOM-VAR

  Ejemplo. Suponiendo que x, y, z son nombres de variables y que ' x + z*y ' pertenece a EXPBF[NOM-VAR], entonces

  Variables(x + z*y) = { x, y, z }

- Dado un conjunto de nombres de variables, la función Ambiente asignará a cada nombre de variable su respectivo valor.

  Ambiente ≡ NOM-VAR --/--> VALOR

  Ejemplo. Suponiendo que el conjunto de nombres de variables es { x, y, z } y que val1, val2, val3 son valores, un Ambiente podría estar definido por

  Ambiente = {x !--> val1, y !--> val2, z !--> val3}

- La evaluación del resultado de una expresión bien formada se obtiene al aplicar la función Evalúa a dicha expresión, suponiendo un ambiente determinado.

  Evalúa : Ambiente --/--> EXPBF --/--> VALOR

  Ejemplo. Dada la expresión y el Ambiente de los ejemplos anteriores

  Evalúa(Ambiente)(x + z*y) = val1 + val3 * val2

- Dada una expresión bien formada la función Tipo-exp proporciona el nombre del dominio al cual debe pertenecer el resultado de la evaluación de dicha expresión.

  Tipo-exp : EXPBF[NOM-VAR] --/--> NOM-DOM

  Ejemplo. Si suponemos que val1,val2, val3 son valores del dominio ENTERO entonces

  Tipo-exp(x + z*y) = ENTERO

A cada categoría de atributo se le podrá asociar un lenguaje de la familia parametrizado por un conjunto de nombres de variables específico que se definirá apropiadamente en cada caso (46).

V.1.2 ATRIBUTOS GENERADOS.

Un atributo generado de una clase es aquel cuya definición es generada por, o está en función de, otros atributos de la misma clase.

V.1.2.1 DEFINICION.

Los atributos generados de una clase serán modelados introduciendo tres nuevos componentes en el esquema DEF-CLASE:

- Una función, llamada AG, que asocia a cada nombre de atributo generado el nombre del dominio al que pertenece su valor.
- Una función, llamada Def-AG que asocia a cada nombre de atributo generado la expresión bien formada que lo define.

  En este caso, las expresiones bien formadas estarán parametrizadas por el conjunto de nombres de variables igual al conjunto de todos los posibles nombres de atributos (NOM-ATR), es decir el lenguaje consistirá del conjunto EXPBF[NOM-ATR], a este lenguaje le llamaremos lenguaje de expresiones de atributos generados.

  Por convensión, al conjunto de expresiones del lenguaje le llamaremos EXP-ATR-GEN, es decir

$$\text{EXP-ATR-GEN} \equiv \text{EXPBF[NOM-ATR]}$$

- Un conjunto Atr-Gen formado por los nombres de todos los atributos generados de la clase. Este componente no es estrictamente necesario (ya que es derivable de los anteriores componentes) pero facilita la referencia a los atributos generados.

Para facilitar la referencia a cualquiera de los atributos de una clase independientemente de la categoría del atributo, incluiremos en la redefinición de DEF-CLASE los componentes derivados:

- La función Tipo-Atr que asocia a cada nombre de atributo su nombre de dominio correspondiente, ya sea aplicando AP o aplicando AG.

- El conjunto Atributos cuyos elementos son todos los nombres de atributos de la clase, sin importar su categoría.

El siguiente esquema redefine DEF-CLASE, incluyendo los componentes mencionados y los invariantes que deberán satisfacer.

EXPBF[NOM-VAR];

EXP-ATR-GEN ≡ EXPBF[NOM-ATR];

DEF-CLASE

DEF-CLASE

AG: NOM-ATR --//--> NOM-DOM

Def-AG: NOM-ATR --//--> EXP-ATR-GEN

Atr-Gen: F NOM-ATR

Tipo-Atr: NOM-ATR --//--> NOM-DOM

Atributos: F NOM-ATR

∀ exp : rng Def-AG .

Variables(exp) C Atributos I15

Atr-Gen = dom AG = dom Def-AG I16

Atr-Gen ∩ Atr-Prim = {} I17

∀ a : Atr-Gen .

AG(a) = Tipo-exp(Def-AG(a)) I18

Tipo-Atr = AP ∪ AG I19

rng Tipo-Atr C dom Val-Dom	I20
Atributos = Atr-Gen U Atr-Prim	I21

I15: Las variables de una expresión que define a un atributo generado deben ser atributos de la misma clase.

I16: Definición del conjunto de nombres de atributos generados.

I17: No se permite llamar a los atributos de distintas categorías de la misma forma.

I18: El dominio definido para un atributo generado es igual al dominio del tipo de expresión que lo define.

I19: Definicion del tipo de un atributo (cualquiera).

I20: Todos los dominios correspondientes a los atributos tienen un conjunto de valores asociado.

I21: Definición del conjunto de nombres de atributos (generales) de la clase.

Nota Técnica.- La función Variables, en el contexto del lenguaje de expresiones de atributos generados, está definida por Variables:EXP-ATR-GEN -/-> F NOM-ATR; formalmente se debería denominar Variables[NOM-ATR] pero se omite el parámetro cuando no da origen a confusión. La misma observación vale para Tipo-exp y en general, para todas las funciones del lenguaje de expresiones.

Ejemplo.

Supongamos, como caso de estudio, a una clase c5 con la siguiente definición, que llamaremos DEF-CLASE-C5.

AP = { atr1 !--> ENTERO, atr2 !--> ENTERO }

Atr-Prim = { atr1, atr2 }

AG = ⟨ atr3 !--> ENTERO, atr4 !--> ENTERO ⟩

Def-AG = ⟨atr3 !--> atr1 + atr2, atr4 !-- atr2 - atr1⟩

donde c5 pertenece a NOM-CLASE; atr1, atr2, atr3, atr4 pertenecen a NOM-ATR; y donde ENTERO pertenece al dominio de Val-Dom, entonces los componentes derivados quedan definidos asi:

Atr-Gen = ⟨atr3, atr4⟩ = dom AG = dom Def-AG (por I16)

Tipo-Atr = ⟨atr1 !--> ENTERO, atr2 !--> ENTERO,
atr3 !--> ENTERO, atr4 !--> ENTERO⟩
= AP U AG (por I19)

Atributos = ⟨atr1, atr2, atr3, atr4⟩
= Atr-Prim U Atr-Gen (por I21)

Además se satisfacen los invariantes restantes ya que

rng Def-AG = ⟨atr1 + atr2, atr2 - atr1⟩

Variables(atr1 + atr2)= ⟨atr1, atr2⟩ C Atributos

Variables(atr2 - atr1)= ⟨atr2, atr1⟩ C Atributos

por lo tanto se satisface I15. Se satisface I17 ya que

Atr-Gen = ⟨atr3, atr4⟩ ∩ ⟨atr1, atr2⟩ = ⟨⟩

Se satisface I18 ya que

AG(atr3) = ENTERO

Tipo-exp(Def-AG(atr3)) = Tipo-exp(atr1 + atr2)
= ENTERO

AG(atr4) = ENTERO

Tipo-exp(Def-AG(atr4)) = Tipo-exp(atr2 - atr1)
= ENTERO

por lo tanto

AG(atr3) = Tipo-exp(Def-AG(atr3)) y

AG(atr4) = Tipo-exp(Def-AG(atr4))

Se satisface I20 ya que

rng Tipo-Atr = ENTERO C dom Val-Dom

La inclusión de los nuevos componentes en DEF-CLASE hace necesaria la actualización del invariante de consistencia en el tipo de los atributos de las instancias (I9):

V (c,id): dom Atr-Inst .

Atr-Inst(c,id) C Esq(c).Atr-Prim

Recordemos que este invariante garantiza que todos los atributos de una instancia hayan sido declarados como atributos primitivos en la definición de clase correspondiente. Puesto que hemos extendido el concepto de atributos incluyendo otra categoría, es necesario garantizar que también los nombres de atributos generados instanciados correspondan con los nombres de atributos generados declarados.

El componente Atributos, ya incorporado a DEF-CLASE, nos será útil en la siguiente actualización de I9.

V (c,id): dom Atr-Inst .

Atr-Inst(c,id) C Esq(c).Atributos (I9)

Puesto que Atributos es el conjunto de todos los atributos de la clase independientemente de su categoría,

ha quedado garantizado que en la instanciación de atributos generados haya correspondencia entre los nombres de atributos instanciados y los declarados.

Es necesaria también otra actualización, la de consistencia en los valores asociados a los atributos de las instancias (I10):

$$\forall\ (c,id,a):\ Valor\text{-}Inst\ .$$
$$Valor\text{-}Inst(c,id,a) \in Val\text{-}Dom\ (Dominio(c,a))$$

Recordemos que este invariante garantiza que los valores de atributos instanciados pertenezcan a los valores de su dominio correspondiente en la definición de la clase.

Requerimos que se cumpla I10 para instancias de atributos generados, los cuales no se consideraban en el invariante anterior, ya que el selector Dominio se definió así

$$Dominio = \{\ (c,a) \mapsto Esq(c).AP(a)\ /\ c: dom\ Esq,\ a: dom\ Esq(c).AP\ \}$$

La siguiente redefinición de Dominio actualiza automáticamente a I10.

```
┌──────────────────────────────────────────────────┐
│                                                  │
│     BDM                                          │
│                                                  │
│     Dominio: NOM-CLASE X NOM-ATR ----> NOM-DOM   │
│     ____________                                 │
│                                                  │
│     Dominio = { (c, a) |---> Esq(c).Tipo-Atr(a)/ │
│                                                  │
│               c: dom Esq, a: dom Esq(c).Tipo-Atr } │
│                                                  │
```

De esta forma, el selector Dominio asociará a cada clase y atributo (de cualquier categoría) el dominio definido para ese atributo, sea primitivo o sea generado, con lo cual I10 garantiza, ahora sí, la consitencia en los valores asociados a atributos de cualquier categoría.

Una observación importante respecto a atributos generados, es que existe la posibilidad de que haya atributos generados definidos en términos de atributos que, a su vez también sean generados. Note que I15 permite que las variables de una expresión que defina a un atributo generado, se refieran a cualquier nombre de atributo de la clase, incluyendo a los generados. Esta situación podría ocasionar que se tuvieran 'ciclos' en la definición de atributos, que imposibilitaran el cálculo de sus valores. Para evitar que en BDM se den estos casos, definiremos un selector, llamado _depende_ , que nos será muy útil en esta tarea.

El selector _depende_ es una relación entre dos atributos (a_i , a_j) de la misma clase. Diremos que

$$a_i \text{ depende } a_j$$

si a_i es un atributo generado y si en la expresión que lo define, aparece el atributo a_j como variable.

```
_depende_ : NOM-ATR <----> NOM-ATR
_____________

a  depende a    <===> (a  ∈ Atr-Gen) &
 i          j          i
                      a  ∈ Variables(Def-AG(a ) )
                       j                    i
```

Nota Técnica.- Puesto que _depende_ es una relación, es posible que un atributo dependa de varios atributos.

Ejemplo.

Tomando nuestro caso de estudio anterior, la clase c5, observamos las siguientes relaciones de dependencia

atr3 depende atr1

atr3 depende atr2

atr4 depende atr1

atr4 depende atr2

La siguiente redefinición de DEF-CLASE incluye un invariante que garantiza que las definiciones de atributos generados de la clase no establecen ciclos de dependencia que ocasionen la indeterminación de sus valores.

```
DEF-CLASE_____________________________________________
DEF-CLASE
_____________
                +
_depende_   ∩  id NOM-ATR = {}                    I22
```

I22: Atributos generados bien definidos.

Nota Técnica.- La notación _depende_$^{+}$ representa la

cerradura transitiva de la relación _depende_ ;
id NOM-ATR es la función identidad sobre NOM-ATR
(para más detalles consulte la introducción a "Z")

Ejemplos.

En nuestro caso de estudio se cumple el invariante I22 ya que

$$\_depende\_ = \{atr3 \mapsto atr1,\ atr3 \mapsto atr2,\ atr4 \mapsto atr1,\ atr4 \mapsto atr2\}$$

$$\_depende\_;\_depende\_ = \{atr3 \mapsto atr1,\ atr3 \mapsto atr2,\ atr4 \mapsto atr1,\ atr4 \mapsto atr2\};\ \{atr3 \mapsto atr1,\ atr3 \mapsto atr2,\ atr4 \mapsto atr1,\ atr4 \mapsto atr2\} = \_depende\_$$

por lo tanto

$$\_depende\_^{+} = \_depende\_$$

y

$$\_depende\_^{+} \cap \{atr1 \mapsto atr1,\ atr2 \mapsto atr2,\ atr3 \mapsto atr3,\ atr4 \mapsto atr4\} = \{\}$$

Vemos que la cerradura transitiva de _depende_ para los atributos generados de la clase c5 es la misma relación ya que su dominio no se intersecta con su rango, por lo que no hay transitividad. Por lo tanto, los atributos generados están bien definidos.

En el siguiente ejemplo observaremos un caso en el cual no se cumple el invariante. Supongamos que una clase tuviera definidos los siguientes atributos generados

Def-AG(atr3) = atr4

Def-AG(atr4) = atr3

Obviamente esta definición es cíclica y no debe permitirse. En este caso resulta que el invariante no se cumple ya que

$$\_depende\_ = \{atr3 \mapsto atr4, atr4 \mapsto atr3\}$$

$$\_depende\_ ; \_depende\_ = \{atr3 \mapsto atr4, atr4 \mapsto atr3\} ; \{atr3 \mapsto atr4, atr4 \mapsto atr3\} = \{atr3 \mapsto atr3, atr4 \mapsto atr4\}$$

$$\_depende\_^{+} = \_depende\_ ; \_depende\_ \cup \_depende\_ = \{atr3 \mapsto atr3, atr4 \mapsto atr4\} \cup \_depende\_ = \{atr3 \mapsto atr3, atr4 \mapsto atr4, atr3 \mapsto atr4, atr4 \mapsto atr3\}$$

entonces

$$\_depende\_^{+} \cap id_{NOM\text{-}ATR} = \{atr3 \mapsto atr3, atr4 \mapsto atr4\} \neq \{\}$$

Por lo tanto I22 no se cumple, y de esta forma, se ha mostrado su utilidad para detectar casos de dependencia cíclica.

V.1.2.2 CONSISTENCIA DE INSTANCIAS.

Hasta el momento, todavía no hemos considerado en el modelo la forma de evaluar las expresiones que definen a los atributos generados. En este aspecto debemos notar que puede generarse un problema cuando, al evaluar dichas expresiones, no se conoce el valor de algun atributo, es decir, no se tiene la información suficiente para evaluar

la expresión, en cuyo caso, el valor del atributo generado correspondiente queda indeterminado.

Para incluir el invariante que garantice la evaluación de los atributos generados en la forma correcta, nos serán muy útiles las funciones Ambiente y Evalúa del lenguaje de expresiones.

En la siguiente redefinición de BDM incluimos los aspectos mencionados y concluimos con la especificación de los atributos generados.

BDM

```
BDM
-----------------
V (c , id, a ) : dom Valor-Inst .
[ (a : Esq(c).Atr-Gen)  ===>
  [ V a  : Variables (Esq(c).Def-AG(a)) .
       i
       a  ∈ dom Ambiente ]
        i
  &
  [ Valor-Inst(c, id, a) =
       Evalúa(Ambiente)(Esq(c).Def-AG(a))]  ]  I23
  Donde
     Ambiente = Inst(c).IC(id)
```

I23: Los valores de las variables de una expresión que define un atributo generado están definidos en el ambiente y el valor de un atributo generado de una instancia de c está dado por la evaluación de la expresión que lo define en el ambiente.

Nota Técnica.- La función Ambiente en el contexto del

lenguaje de atributos generados está definida como Ambiente = NOM-ATR -/-> VALOR = Inst(c).IC(id), donde c es una clase definida,y donde id en un identificador de instancia de c.

Ejemplo.

Supongamos que tenemos una BDM con una sola clase definida, nuestro caso de estudio c5, la cual tiene dos instancias, EBDM tiene como componente a

Esq = { c5 !--> DEF-CLASE-C5 }

IBDM está definido por

Inst(c5).IC = {id1 !--> {atr1 !--> 10, atr2 !--> 15,
atr3 !--> 25, atr4 !--> 5} ,
id2 !--> {atr1 !--> 25, atr2 !--> 30,
atr3 !--> 55, atr4 !--> 5} }

Inst(c5).Instancias = { id1, id2 }

Entonces

(c5, id1, atr3)! dom Valor-Inst

ya que c5!dom Inst, id1! dom Inst(c5).IC
y atr1! dom Inst(c5).IC(id1) (consulte la definición del selector Valor-Inst)

Ambiente = Inst(c5).IC(id1)

= {atr1 !--> 10, atr2 !--> 15,
atr3 !--> 25, atr4 !--> 5}

Variables(Esq(c5).Def-AG(atr3)) = {atr1, atr2}

por lo tanto se cumple que

atr1, atr2 ∈ dom Ambiente

y

Valor-Inst(c5, id1, atr3) =

Evalúa(Ambiente)(Esq(c5).Def-AG(atr3)) =

Evalúa(Ambiente)(atr1 + atr2) = 10 + 15 = 25

Lo mismo puede verificarse para (c5,id1,atr4), para

(c5,id2,atr3) y (c5,id2,atr4) con lo cual se satisface
(47)
I23 .

V.I.3 ATRIBUTOS HEREDADOS.

En esta sección introducimos en el modelo otra categoría de atributos, a los cuales hemos llamado atributos heredados. Como su nombre lo indica, estos atributos son aquellos que heredan valores de atributos de otras clases. Mediante la definición de ellos es posible establecer mecanismos muy diversos de herencia y tipos de jerarquías flexibles.

Si una clase va a tener atributos heredados, entonces en su definición debe especificarse toda la información necesaria para que no haya ambigüedad en la determinación de los valores que se van a heredar.

Además, en la especificación formal de los atributos heredados deben considerarse varias restricciones para que la herencia pueda efectuarse. Paulatinamente avanzaremos en la inclusión del concepto en el modelo, empezando por una nueva redefinición de DEF-CLASE.

V.1.3.1 DEFINICION.

Los atributos heredados serán incluídos en BDM introduciendo, en primer lugar, tres nuevos componentes en DEF-CLASE (análogos a los que incluimos para atributos generados):

- Una función, llamada AH, que asocia a cada atributo heredado el dominio al que pertenece su valor.
- Una función, llamada Def-AH, que asocia a cada atributo heredado la expresión bien formada que lo define.

  En el caso de atributos heredados, los nombres de las variables que parametrizan a cada expresión bien formada deberán ser del tipo que hemos llamado VAR (la especificación de este conjunto se presenta junto con la redefinición de DEF-CLASE). Al lenguaje de expresiones bien formadas parametrizado por el conjunto de nombres de variables VAR le llamaremos lenguaje de expresiones de atributos heredados. Las expresiones de este lenguaje serán llamadas EXP-ATR-HER y se cumple que

$$EXP\text{-}ATR\text{-}HER \equiv EXPBF[VAR]$$

- Un conjunto Atr-Her formado por los nombres de los atributos heredados de la clase. Este componente derivable facilita la referencia al conjunto de atributos heredados.

En la definición de atributos generados, se introdujeron dos componentes más: Tipo-Atr y Atributos para poder hacer referencia a cualquier atributo, independientemente de su categoría. En este caso, esos mismos componentes nos serán útiles, únicamente será necesario actualizar sus respectivos invariantes, el que define el tipo de un atributo cualquiera (I19) y el que define el conjunto de nombres de atributos de la clase (I21) incluyendo a los atri-

```
Def-AH: NOM-ATR --//--> EXP-ATR-HER
Atr-Her: F NOM-ATR
```

```
Atr-Prim  ∩ Atr-Gen ∩ Atr-Her = {}                    I17
Tipo-Atr = AP ∪ AG ∪ AH                               I19
Atributos = Atr-Prim ∪ Atr-Gen ∪ Atr-Her              I21
Atr-Her = dom AH = dom Def-AH                         I24
∀ a: Atr-Her  .
     AH(a) = Tipo-exp(Def-AH(a))                      I25
```

I24: Definición del conjunto de atributos heredados.

I25: El dominio definido para un atributo heredado es igual al dominio del tipo de expresión que lo define.

Nota Técnica.- La función Tipo-exp, en el contexto del lenguaje de expresiones de atributos heredados, tendrá como signatura EXP-ATR-HER ---> NOM-DOM, formalmente se debería llamar Tipo-exp[VAR] pero se omite el parámetro pues no da origen a confusión. La misma observación vale para las demás funciones del lenguaje de expresiones de atributos heredados.

Ejemplo.

Tomemos la clase c5 del apartado anterior. Suponga-mos que deseamos definir dos atributos heredados, entonces en DEF-CLASE-C5 incluiremos los siguientes componentes.

```
AH = {atr5 !--> ENTERO, atr6 !--> ENTERO}
Def-AG = {atr5 !--> var1 + var2, atr6 !--> var1}
Atr-Her = {atr5, atr6}
Tipo-Atr = {atr1 !--> ENTERO, atr2 !--> ENTERO,
            atr3 !--> ENTERO, atr4 !--> ENTERO,
            atr5 !--> ENTERO, atr6 !--> ENTERO}
```

Atributos = {atr1, atr2, atr3, atr4, atr5, atr6}

donde atr5, atr6 son nombres de atributos y donde var1 y var2 son de tipo VAR y están definidas por

var1.Atributo.Cla = c6

var1.Atributo.Atr = atr1

var1.Rol = r1

var2.Atributo.Cla = c7

var2.Atributo.Atr = atr1

var2.Rol = r2

donde suponemos que c6, c7 son clases definidas, atr1 ∈ Esq(c6).Atributos, r1 ∈ Esq(c6).Roles y donde atr1 ∈ Esq(c7).Atributos, r2 ∈ Esq(c7).Roles (estos requerimientos no se han especificado aun, pero será necesario más adelante).

El significado intuitivo de la expresión

Def-AH(atr5) = var1 + var2

es la suma del atributo atr1 de c6 (a través del rol r1) más el atributo atr1 de c7 (a través del rol r2). El atributo heredado atr6 será igual al atributo atr1 de c6 (a través del rol r1).

Puede comprobarse, en forma análoga a como se mostró para atributos generados, que se satisfacen los invariantes I17, I19, I21, I24 e I25.

La introducción del concepto de atributos heredados no ocasionará, como en el caso de los atributos generados,

una actualización de los invariantes de consistencia en los nombres de atributos instanciados (I9) y de consistencia en los valores asociados a los atributos de las instancias (I10), ya que la inclusión de los componentes derivados Tipo-Atr y Atributos en DEF-CLASE nos permitió la generalización de dichos invariantes al actualizar, incluyendo a los atributos heredados, a I19 e I21 del esquema anterior (49).

Podemos observar que en los predicados del esquema DEF-CLASE que introduce los atributos generados, el primer invariante se refiere a requisitos que las variables de las expresiones deben satisfacer (I15). Es necesario incluir un invariante análogo para las variables de las expresiones que definan atributos heredados. Los requisitos que dichas variables deben satisfacer son los que se piden en el ejemplo anterior para var1 y var2. Para especificar estos aspectos definiremos los siguientes selectores:

- ATRIBUTOS, para denotar el conjunto de todos los nombres de atributos definidos en EBDM (de todas las clases). Note que no es posible hacer referencia a los atributos sin referirse a la clase a la cual pertenecen (esto sería ambiguo ya que pueden repetirse los nombres de atributos en diferentes clases), por lo cual se utilizan en la definición del selector las variables de tipo ATRIB, que nos auxilian en la notación al representar siempre al atributo junto con la clase a la que pertenece.
- La función Clase que asocia a una variable de tipo VAR

la clase que denota, y la función Atrib que le asocia el atributo al que se refiere. Estas funciones sirven sólo para simplificar la notación.

Los selectores quedan definidos por los siguientes esquemas.

```
EBDM
ATRIBUTOS: F ATRIB
-----------------
ATRIBUTOS = { a: ATRIB/ a.Cla ∈ dom Esq  ∧
                    a.Atr ∈ Esq(a.Cla).Atributos}
```

Nota Técnica.- Observar que el selector ATRIBUTOS que se define en este esquema en un conjunto de elementos de tipo ATRIB. No confundir con el campo Atributos que es el conjunto de nombres de atributos de una clase. La misma observación vale para todos los componentes de esquemas o funciones que se llamen de la misma forma pero variando en minúsculas y mayúsculas.

```
EBDM
Clase : VAR ---> NOM-CLASE
Atrib : VAR ---> NOM-ATR
-----------------
∀ v : VAR  .
     Clase(v) = v.Atributo.Cla
     Atrib(v) = v.Atributo.Atr
```

En el siguiente esquema se incluye la especificación de los requisitos necesarios que debe satisfacer cada variable v de una EXP-ATR-HER:

a) Que Clase(v) sea una clase definida en el EBDM.

b) Que Atrib(v) sea un atributo de esa clase.

c) Que v.Rol sea un rol que pertenezca al conjunto de roles de Clase(v).

EBDM

EBDM	
∀ c : dom Esq ; exp : rng Def_AH; v : Variables (exp) . v.Atributo ∈ ATRIBUTOS & v.Rol ∈ Esq(Clase(v)).Roles	I26

I26: Toda variable de una expresión que define un atributo heredado se refiere a clases definidas, a atributos de esas clases y a roles de ellas.

Nota Técnica.- Que v.Atributo ∈ ATRIBUTOS implica (por la definición de selectores ATRIBUTOS, Clase y Atrib) que Clase(v) ∈ dom Esq, y que Atrib(v) ∈ Esq(Clase(v)).Atributos. Eso implica a su vez a) y b). Que v.Rol ∈ Esq (Clase(v)).Roles implica c).

Ejemplo.

En el ejemplo anterior hicimos varias suposiciones respecto a c6 y c7. Daremos aquí sus definiciones, para

después checar que se satisface I26 en un EBDM cuyo dominio contenga a c5, c6 y c7.

Definición de c6:

Esq(c6).AP = {atr1 !--> ENTERO}

Esq(c6).Atributos = {atr1}

Esq(c6).Agreg-Rol = { r1 !--> {c5} }

Esq(c6).Agregados = {c5}

Esq(c6).Roles = {r1}

Definición de c7:

Esq(c7).AP = {atr1 !--> ENTERO}

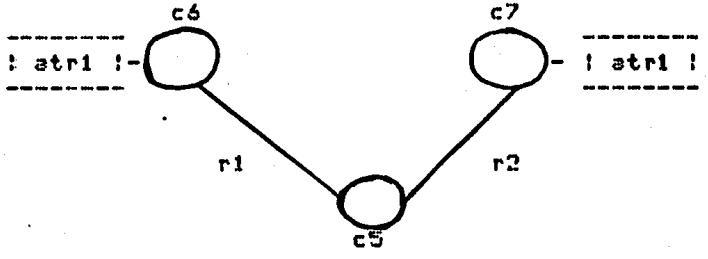
Esq(c7).Atributos = {atr1}

Esq(c7).Agreg-Rol = { r2 !--> {c5} }

Esq(c7).Agregados = {c5}

Esq(c7).Roles = {r2}

La gráfica que mostraria la situación en EBDM sería la siguiente.



Recordemos que en el ejemplo anterior se definieron dos atributos heredados de la siguiente manera.

Def-AH(atr5) = var1 + var2

Def-AH(atr6) = var1

donde las definiciones de var1 y var2 estaban dadas por

Clase(var1) = c6

Atrib(var1) = atr1

var1.Rol = r1

Clase(var2) = c7

Atrib(var2) = atr1

var2.Rol = r2

Note que hemos utilizado aquí los selectores Clase y Atrib para simplificar la notación.

Vemos que entonces, se cumple que

var1.Atributo ∈ ATRIBUTOS

ya que

var1.Atributo.Cla=Clase(var1)=c6 ∈ dom Esq

y

var1.Atributo.Atr=Atrib(var1)=atr1 ∈ Esq(c6).Atributos

También se cumple, por argumentos similares, que

var2.Atributo = ∈ ATRIBUTOS

Además

var1.Rol = r1 ∈ Esq(c6).Roles

var2.Rol = r2 ∈ Esq(c7).Roles

por lo tanto, toda variable de una expresión que define un atributo heredado se refiere a clases definidas, a atributos de esas clases y a roles de ellas, es decir, se satis-

face el invariante I26.

Este ejemplo también puede servir al lector para observar que los atributos atr5, atr6 de c5 heredan de clases (c6 y c7) que tienen entre sus agregados a c5. Este requisito será necesario en general para que se pueda dar la herencia , será formalizado más adelante.

Para evitar ciclos de dependencia en las definiciones de atributos heredados, que ocasionen la indeterminación de sus respectivos valores, definiremos un selector, al que denominamos _depende-Her_ , análogo al definido para atributos generados, que nos auxiliará en este aspecto de la especificación.

Definiremos el selector _depende-Her_ como una relación entre dos atributos.

Nos interesa que este selector relacione un atributo heredado con aquellos atributos de los cuales depende, es decir, con los atributos que aparecen en su definición. Tomando en cuenta que el atributo heredado pertenecerá a una clase determinada, y los atributos que aparezcan en la expresión que lo define pertenecerán a otra clase, definiremos al selector como una relación entre elementos del conjunto previamente definido ATRIBUTOS, lo que nos permitirá hacer referencia al atributo y a su clase con una misma variable (de tipo ATRIB).

Una pareja a_i , a_j de tipo ATRIB estará en la relación _depende-Her_ si el atributo de a_i (a_i.Atr) está definido como heredado, y si alguna de las variables de

la expresión que lo define se refiere al atributo de a_j (a_j.Atr) y a su misma clase (a_j.Cla).

BDM

depende-Her : ATRIBUTOS <---> ATRIBUTOS

---

a_i depende-Her a_j <==>

(a_i.Atr : Esq (a_i.Cla).Atr-Her)

%

(∃ v : Variables (Esq(a_i.Cla).Def-AH(a_i.Atr)) •

a_j.Cla = Clase(v)

%

a_j. Atr = Atrib(v))

Ejemplo.

Tomemos nuestro caso de estudio, la clase c5 ya definida, donde

Def-AH = {atr5 !--> var1 + var2, atr6 !--> var1}

Atr-Her = {atr5, atr6}

Clase(var1) = c6

Atrib(var1) = atr1

var1.Rol = r1

Clase(var2) = c7

Atrib(var2) = atr1

var2.Rol = r2

Entonces, si a_i está dada por

$a_i.Cla = c5$, $a_i.Atr = atr5$

y a_j está dada por

$a_j.Cla = c7$, $a_j.Atr = atr1$

podemos afirmar que

a_i depende-Her a_j

ya que

(atr5 ∈ Esq(c5).Atr-Her) &

(∃ var2: Variables(Esq(c5).Def-AH(atr5)) .

$a_j.Cla = c7 = Clase(var2)$

$a_j.Atr = atr1 = Atrib(var2)$)

El lector puede comprobar, de la misma forma, que si

$a_j.Cla = c6$ y $a_j.Atr = atr1$ entonces también

a_i depende-Her a_j

y que lo mismo se cumple cuando $a_i.Cla = c5$ y $a_i.Atr = atr6$.

En la siguiente redefinición de BDM incluiremos un invariante, análogo a I22 (atributos generados bien definidos) que garantizará que los atributos heredados no sean dependientes cíclicamente, evitando así que estén mal definidos (50).

EDM

EDM	
$\_depende\text{-}Her\_^{+}$ $\cap$ id ATRIBUTOS = $\{\}$	I27

I27: Atributos heredados bien definidos.

Ejemplos.

Puede comprobarse que si EDDM tiene definidas solo a las clases c5, c6, y c7, entonces

$$\_depende\text{-}Her\_^{+} \cap id\ ATRIBUTOS = \{\}$$

La forma de probarlo es completamente análoga al ejemplo que ilustra la satisfacción del invariante I22.
Puesto que se satisface I27, no existe dependencia cíclica y por lo tanto, los atributos heredados están bien definidos.

Un ejemplo de atributos heredados dependientes cíclicamente es el siguiente.

Supongamos que las clases c, c', tienen definidos los siguientes componentes.

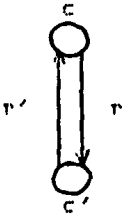
Esq(c).Agreg-Rol = { r !--> {c'} }

Esq(c).Def-AH = {atrH !--> atrH'}

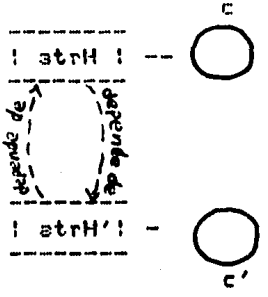
Esq(c').Agreg-Rol = { r' !--> {c} }

Esq(c').Def-AH = {atrH' !--> atrH}

En este caso, c tiene como agregado a través de r a c'; a su vez, c' tiene como agregado a través de r' a c, lo cual es válido (no hay ningún invariante que impida esta situación y se representa en la siguiente gáfica.



Sin embargo, los atributos heredados son dependientes cíclicamente y no podrá determinarse su valor.



Puede probarse que I27 no se satisface y por lo tanto, los atributos heredados atrH y atrH' no están bien definidos.

En todos los ejemplos de definición de atributos heredados que hemos visto hasta el momento, solo se han presentado casos de herencia de atributos de clases a agregados de ellas (es decir a un nivel) aunque este requerimiento aun no ha sido especificado formalmente.

Consideramos conveniente establecer esta limitación en la herencia para evitar complicar más el modelo. Por otro lado, la capacidad del modelo no se restringirá a la herencia a un nivel ya que es posible establecer cadenas de herencia para lograr que un atributo se herede hasta el nivel que se desee, a apartir de la herencia a un nivel. Es decir, una clase puede heredar a un agregado, éste también podrá heredar a un agregado el mismo atributo, y así sucesivamente hasta el nivel deseado (51).

En el siguiente esquema introduciremos el invariante que formaliza los requisitos para herencia a un nivel. En él se establece que la clase c a la que pertence un atributo heredado debe estar en el conjunto de agregados de las clases a las que hacen referencia las variables de la expresión que define el atributo heredado.

Es decir, si v es una variable de dicha expresión, entonces, c debe pertenecer a Clase(v).Agreg-Rol(v.Rol), lo cual implica que c debe pertenecer al conjunto de agregados de Clase(v) (52).

ERDM

ERDM
∀ c: dom Esq; exp: rng Esq(c).Def-AH;
v: Variables(exp) .
c ∈ Esq(Clase(v)).Agreg-Rol(v.Rol) I28

I28: Las variables que estén en expresiones que defi-

nen atributos heredados hacen referencia a clases de objetos "padre" con roles bien definidos para esa clase (herencia a un nivel).

Ejemplo.

Al revisar las clases definidas c5, c6 y c7, observamos que se cumple el invariante anterior ya que

Variables(Esq(c5).Def-AH(atr5)) = {var1, var2}

Variables(Esq(c5).Def-AH(atr6)) = {var1}

Para var1 se tiene que

c5 ∈ Esq(Clase(var1)).Agreg-Rol(var1.Rol) =

Esq(c6).Agreg-Rol(r1)

Para var2 se tiene que

c5 ∈ Esq(Clase(var2)).Agreg-Rol(var2.Rol) =

Esq(c7).Agreg-Rol(r2)

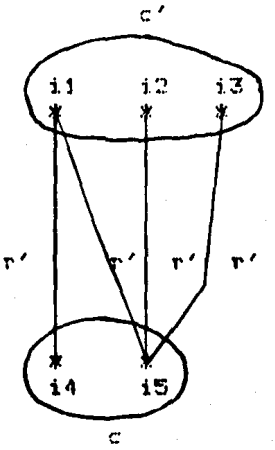
V.1.3.2 CONSISTENCIA DE INSTANCIAS.

Una de las razones que justifican la modificación anterior a EBDM, donde se incluye el invariante de herencia a un nivel, es la necesidad de garantizar la consistencia de instancias cuando se tienen atributos heredados.

Al considerar aspectos de la evaluación de expresiones que definen a atributos heredados, se identifica un problema muy importante: la necesidad de evitar ambigüedad

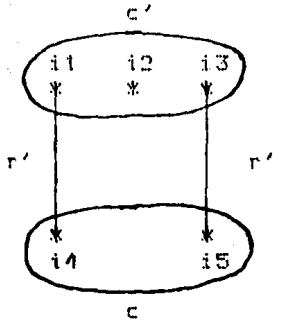
en la instanciación de atributos heredados.

Supongamos que tenemos dos clases cuyas instancias se encuentran agregadas de la siguiente forma.



Si la clase c tiene atributos heredados que dependen de atributos de c', entonces, la evaluación de los atributos heredados de i4 no significará ningún problema. Sin embargo, la evaluación de los atributos heredados de i5 es problemática, ya que existe ambigüedad. ¿De qué instancia heredará i5 los valores de atributos? ¿ De i1, i2 o de i3 ?

Para evitar esta ambigüedad es necesario que las asociaciones entre instancias sean inyectivas (es decir que Mapeo sea inyectiva). Por ejemplo, así.



En general, se garantiza la ausencia de ambigüedad en la instanciación de atributos heredados con el siguiente invariante.

Para toda clase c, que tenga atributos heredados y para toda variable v del conjunto de variables de las expresiones que definen a dichos atributos, debe cumplirse que el Mapeo aplicado a la terna: Clase(v) (igual a c' en el ejemplo anterior), v.Rol y c sea inyectivo.

Este invariante se introduce formalmente en las siguiente redefinición de BDM.

BDM

BDM
∀ c : dom Esq ; exp : rng Esq(c).Def-AH;
v: Variables (exp) .
Mapeo(Clase(v),v.Rol,c) ∈
ID-INST >--//--> ID-INST I29

I29: Ausencia de ambigüedad en instanciación de atributos heredados (Mapeos inyectivos).

Ejemplo.

Supongamos que tenemos una BDM cuyas únicas clases definidas son c y c' (según la última gráfica). Si

Esq(c').Atributos = {atri'} = Esq (c').Atr-Prim

Esq(c).Def-AH = { atri |--> var1 }

donde

Clase(var1) = c'

Atrib(var1) = atri'

var1.Rol = r'

y si

Mapeo(Clase(var1),var1.Rol,c) = Mapeo(c', r', c)

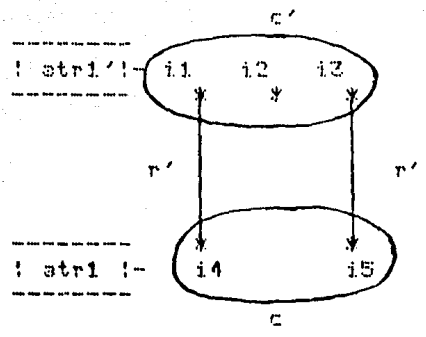
=<i1 |--> i4,i3 |--> i5>

donde i1, i3, i4, i5 pertenecen a ID-INST, entonces

Mapeo(Clase(var1), var1.Rol, c) ∈ (ID-INST >-//->ID-INST)

y por lo tanto se satisface I29.

La ausencia de ambigüedad nos permitirá establecer cómo se lleva a cabo la evaluación de atributos heredados. Tomemos como caso de estudio el ejemplo anterior cuya gráfica es



Sabemos que el valor de un atributo de una instancia de una clase está dado por el selector Valor-Inst. Si deseamos saber que valor tiene el atributo heredado atr1 de la instancia i4 de c, entonces, lo que buscamos es saber a qué es igual Valor-Inst(c,i4,Atr1).

Puesto que contamos con una función de evaluación para el lenguaje de expresiones de atributos heredados entonces

Valor-Inst(c,i4,atr1) = Evalúa(Ambiente)(exp-AH)

= Evalúa(Ambiente)(var1)

donde exp-AH = Esq(c).Def-AH(atr1) = var1

El problema se reduce así a determinar cual es el Ambiente necesario en este caso. El Ambiente para lenguajes de expresiones de atributos heredados esta definido por

Ambiente: VAR --/--> VALOR

dom Ambiente = Variables(exp-AH)

= var1

Para determinar el valor que asocia el Ambiente a var1, aplicamos la función Valor-Inst en la forma siguien-

te.

$$Ambiente(var1) = Valor\text{-}Inst(Clase(var1), id^{-1}(var1), Atrib(var1))$$

Donde $id^{-1}(var1)$ será el identificador al que está asociada la instancia i4, este identificador será una instancia de clase Clase(var1). En nuestro ejemplo, $id^{-1}(var1)$ es igual a i1 (observe la gráfica anterior) (53). Formalmente $id^{-1}(var1)$ se obtiene así

$$\begin{aligned} id^{-1}(var1) &= Mapeo^{-1}(Clase(var1), var1.Rol, c)(i4) \\ &= Mapeo^{-1}(c', r', c)(i4) \\ &= i1 \end{aligned}$$

$Mapeo^{-1}$ es la función inversa de Mapeo, se garantiza su existencia por I29, que pide que Mapeo sea inyectiva, lo cual implica que existe su inversa.

En la siguiente redefinición de BDM se formaliza y generaliza la evaluación de atributos heredados, introduciendo el invariante correspondiente.

BDM

BDM

∀ (c, id, a) : dom Valor-Inst .

(a ∈ Esq(c).Atr-Her) ⟹

[∀ v: Variables(exp-AH) ⟹

$(Clase(v), id^{-1}(v), Atrib(v)) \in$ dom Valor-Inst

&

[Valor-Inst(c,id,a) = Evalúa(Ambiente)(exp-AH)]] I30

Donde

Ambiente: VAR --//--> VALOR

dom Ambiente = Variables(exp-AH)

$\forall$ v: dom Ambiente

Ambiente(v) =

Valor-Inst(Clase(v), id^{-1}(v), Atrib(v))]

Donde

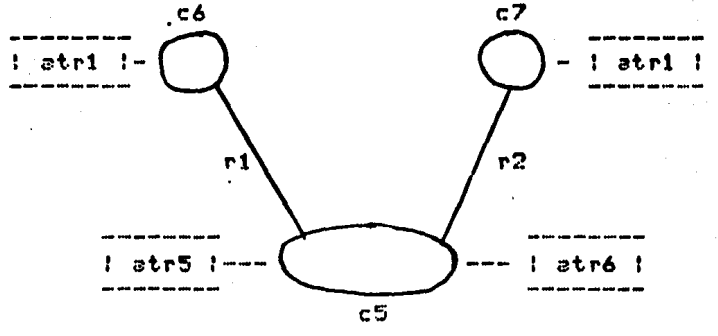
exp-AH = Esq(c).Def-AH(a)

id^{-1}(v) = $Mapeo^{-1}$(Clase(v), v.Rol, c)(id),

I30: El valor de un atributo heredado de una instancia de c está dado por la evaluación que lo define en el ambiente creado (Evaluación de atributos heredados).

Ejemplo.

Tomaremos como ejemplo la BDM cuyo esquema tiene definidas a las clases c5, c6 y c7. Recuerde que la gráfica de esas clases es



Nos interesa también recordar que

Esq(c5).AH = ⟨atr5 !--> ENTERO, atr6 !--> ENTERO⟩

Esq(c5).Def-AH= ⟨atr5 !--> var1 + var2,atr6 !--> var1⟩

Donde

Clase(var1) = c6

Atrib(var1) = atr1

var1.Rol = r1

Clase(var2) = c7

Atrib(var2) = atr1

var2.Rol = r2

y donde

Esq(c6).AP = ⟨atr1 !--> ENTERO⟩

Esq(c6).Agreg-Rol = ⟨ r1 !--> ⟨c5⟩ ⟩

Esq(c7).AP = ⟨atr1 !--> ENTERO⟩

Esq(c7).Agreg-Rol = ⟨ r2 !--> ⟨c5⟩ ⟩

Supongamos que

Inst(c5).IC = ⟨ id1 !--> ⟨atr5 !-->17, atr6 !-->1⟩ ,
id2 !--> ⟨atr5 !-->12, atr6 !-->10⟩ ⟩

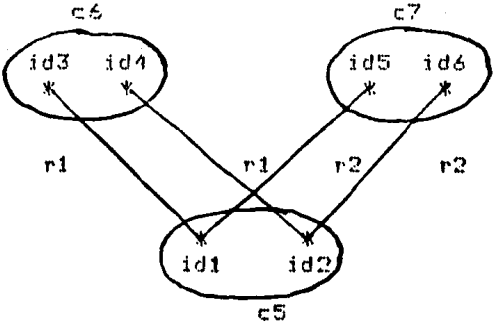
Inst(c6).IC = ⟨ id3 !--> ⟨atr1 !--> 1⟩ ,
id4 !--> ⟨atr1 !--> 10⟩ ⟩

Inst(c7).IC = ⟨ id5 !--> ⟨atr1 !--> 16⟩ ,
id6 !--> ⟨atr1 !--> 2⟩ ⟩

Mapeo(c6,r1,c5) = ⟨ id3 !--> id1, id4 !--> id2 ⟩

Mapeo(c7,r2,c5) = ⟨ id5 !--> id1, id6 !--> id2 ⟩

La gráfica de instancias sería



Verificaremos que se cumple I30 para (c5,id1,atr5).

(c5,id1,atr5)! dom Valor-Inst

atr5 ∈ Esq(c5).Atr-Her

exp-AH = Esq(c5).Def-AH(atr5) = var1 + var2

dom Ambiente = Variables(exp-AH)

= Variables(var1 + var2)

= {var1, var2}

∀ v ∈ dom Ambiente .

$Ambiente(v) = Valor\text{-}Inst(Clase(v), id^{-1}(v), Atrib(v))$

como

$id^{-1}(var1) = Mapeo^{-1}(Clase(var1), var1.Rol, c5)(id1)$

$= Mapeo^{-1}(c6, r1, c5)(id1)$

= id3

y

$id^{-1}(var2) = Mapeo^{-1}(Clase(var2), var2.Rol, c5)(id1)$

$= Mapeo^{-1}(c7, r2, c5)(id1)$

= id5

y además

$(Clase(var1), id^{-1}(var1), Atrib(var1)) = (c6,id3,atr1)$

$\in$ dom Valor-Inst

$(Clase(var2), id^{-1}(var2), Atrib(var2)) = (c7,id5,atr1)$

$\in$ dom Valor-Inst

entonces

Ambiente(var1) = Valor-Inst(c6,id3,atr1) = 1

Ambiente(var2) = Valor-Inst(c7,id5,atr1) = 16

por lo tanto

Valor-Inst(c5,id1,atr5) = Evalúa(Ambiente)(exp-AH)

= Evalúa(Ambiente)(var1+var2)

= 1 + 16 = 17

Lo mismo puede verificarse para (c5,id2,atr5), (54)
(c5,id1,atr6) y (c5,id2,atr6) con lo cual se satisface I30.

V.1.4 ATRIBUTOS RETRIBUIDOS.

En esta sección introduciremos en el modelo una categoría más de atributos, que hemos llamado atributos retribuidos.

Un atributo retribuido de una clase es aquel que está en función de algún o algunos atributos de sus agregados. Es decir, el valor de un atributo retribuido está determinado por atributos de agregados que le retribuyen sus valores.

Los atributos retribuidos constituyen otro de los mecanismos para establecer jerarquías en forma flexible. La retribución es un concepto similar al de herencia, por lo cual abordaremos su especificación en una forma análoga a la de la sección anterior. La diferencia principal entre atributos heredados y retribuidos es que éstos últimos dependen de atributos de agregados o "hijos", mientras que los atributos heredados dependen, como hemos visto, de "padres".

La convinación y uso adecuado de atributos generados heredados y retribuidos proporciona una gran variedad de posibilidades para establecer jerarquías y reglas de herencia muy deversas, de ahí la importancia de estos conceptos.

V.1.4.1 DEFINICION.

Para la inclusión del concepto de atributos retribuidos en el modelo empezaremos redefiniendo una vez más el esquema DEF-CLASE incluyendo los siguientes componentes:

- Una función, que llamaremos AR, la cual asocia a cada atributo retribuido el dominio al que pertenece su valor.
- Una función, que llamaremos Def-AR, la cual asocia a cada atributo retribuido la expresión bien formada que lo define.

  Los nombres de las variables de las expresiones bien formadas serán del mismo tipo (VAR) que los usados para definir atributos heredados. En el caso de atributos retribuidos de una clase c, dado un nombre de variable v de tipo VAR, el componente Clase(v) indicará la clase del atributo que se retribuirá, el componente v.Rol indicará el rol de c a través del cual se retribuirá, y el componente Atrib(v) indicará el atributo de Clase(v) que será retribuido.

  Al lenguaje parametrizado por VAR que usaremos para definir atributos retribuidos le llamaremos lenguaje de expresiones de atributos retribuidos. Al conjunto de expresiones de este lenguaje le llamaremos EXP-ATR-RET, es decir

$$EXP\text{-}ATR\text{-}RET \equiv EXP^{BF}[VAR]$$

  Note que

EXP-ATR-RET ≡ EXP-ATR-HER

- Un componente derivable Atr-Ret que es el conjunto de nombres de atributos retribuidos de la clase.

Además de los componentes anteriores, se incluirán como predicados de DEF-CLASE dos nuevos invariantes: uno que define el conjunto Atr-Ret (I31) y otro que especifica la coincidencia entre el dominio definido para un atributo retribuido y el dominio al que pertenece el valor de la expresió que define al atributo. También se incluirán las actualizaciones de tres invariantes: el que establece que no se permiten nombres iguales para atributos de distintas categorías en la misma clase (I17), el invariante que define el tipo de un atributo cualquiera (I19) y el que define el conjunto de nombres de atributos de una clase (I21).

La definición del conjunto de todas las posibles expresiones del lenguaje de atributos retribuidos y la redefinición del esquema DEF-CLASE que introduce atributos retribuidos son las siguientes.

EXP-ATR-RET ≡ EXP$F[VAR];

DEF-CLASE

DEF-CLASE
AR: NOM-ATR --//--> NOM-DOM
Def-AR: NOM-ATR --//--> EXP-ATR-RET
Atr-Ret: F NOM-ATR

Atr-Prim ∩ Atr-Gen ∩ Atr-Her ∩ Atr-Ret = {}	I17
Tipo-Atr = AP ∪ AG ∪ AH ∪ AR	I19
Atributos =	
Atr-Prim ∪ Atr-Gen ∪ Atr-Her ∪ Atr-Ret	I21
Atr-Ret = dom AR = dom Def-AR	I31
∀ a: Atr-Ret .	
AR(a) = Tipo-exp (Def-AR(a))	I32

I31: Definición del conjunto de atributos retribuidos.

I32: El dominio definido para un atributo retribuido es igual al dominio del tipo de expresión que lo define.

Nota Técnica.- La función Tipo-exp, en el contexto del lenguaje de expresiones de atributos retribuidos, tiene como signatura EXP-ATR-RET -/-> NOM-DOM, formalmente se debería llamar Tipo-exp[VAR] pero omitimos el parámetro para ésta y las demás funciones del lenguaje de expresiones.

Ejemplo.

Supongamos que en la clase c5, que hemos manejado en ejemplos anteriores, deseamos definir un atributo retribuido, entonces en Esq(c5) incluiremos los siguientes componentes.

AR = {atr7 !--> ENTERO}

DEF-AR = {atr7 !--> var3 + var4}

Atr-Ret = {atr7}

Donde atr7 es un nombre de atributo, var3 y var4 son de tipo VAR y están definidas por

Clase(var3) = c8

Atrib(var3) = atr2

var3.Rol = r1

Clase(var4) = c9

Atrib(var4) = atr1

var4.Rol = r1

Supondremos que c8 y c9 son clases definidas, que atr2 ∈ Esq(c8).Atributos, r1 ∈ Esq(c5).Roles, atr1 ∈ Esq(c9).Atributos y r1 ∈ Esq(c5).Roles (estos requisitos no se han especificado aun, pero será necesario hacerlo más adelante).

La expresión

Def-AR(atr7) = var3 + var 4

significa que el atributo atr7 será igual a la suma del atributo atr2 de c8 a través del rol r1 más el atributo atr1 de c9 a través de r1.

Fácilmente, pueden calcularse los componentes Tipo-Atr y Atributos según se establece en I19 e I21 . Además puede comprobarse que se satisfacen I17, I24 e I25.

La consistencia en los nombres de atributos instanciados (I9) y la consistencia en los valores asociados a los atributos de las instancias (I10) para el caso de atributos retribuidos ha quedado garantizada automáticamente con la inclusión de atributos retribuidos en Tipo-Atr (I19) y en Atributos (I21). Se recomienda al

lector comprobar esto.

Cada variable v de una expresión de atributos retribuídos de una clase c debe satisfacer tres requisitos (como los que se piden a var3 y var4 en el ejemplo anterior):

a) Que Clase(v) sea una clase definida.

b) Que Atrib(v) sea un atributo de esa clase.

c) Que v.Rol sea un rol que pertenezca al conjunto de roles de c (55).

La especificación de estos aspectos se incluirá en la siguiente redefinición de EBDM.

EBDM

EBDM

∀ c : dom Esq ; exp : rng Def_AR ∣

v : Variables (exp) .

v.Atributo ∈ ATRIBUTOS **I33**

&

v.Rol ∈ Esq(c).Roles

I33: Toda variable de una expresión que define un atributo retribuido se refiere a clases definidas, a atributos de esas clases y a roles de la clase en la que se define el atributo.

Nota Técnica.- Que v.Atributo ∈ ATRIBUTOS implica (por la definición de los selectores ATRIBUTOS, Clase y Atrib) que Clase(v) ∈ dom Esq y que Atrib(v) ∈ Esq(Clase(v)).Atributos. Eso implica a su vez a) y b). Que v.Rol∈Esq(c).Roles implica c) (56).

Ejemplo.

Hemos supuesto que c8 y c9 son clases definidas. Supongamos ahora que Esq(c8) y Esq(c9) tienen los siguientes componentes.

Esq(c8).AP = {atr2 !--> ENTERO}

Esq(c8).Atributos = {atr2}

Esq(c9).AP = {atr1 !--> ENTERO}

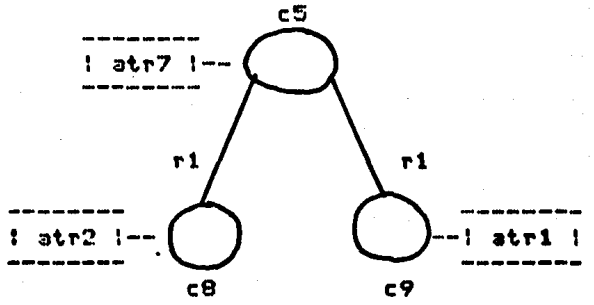
Esq(c9).Atributos = {atr1}

Supongamos también que

Esq(c5).Agreg-Rol = { r1 !--> {c8,c9} }

Esq(c5).Roles = {r1}

La gráfica que mostraría la situación de EBDM sería



donde el atributo retribuido atr7 se define por

Def-AR(atr7) = var3 + var4

y donde

Clase(var3) = c8

Atrib(var3) = atr2

var3.Rol = r1

Clase(var4) = c9

Atrib(var4) = atr1

var4.Rol = r1

Entonces se cumple que

Clase(var3)=c8 ∈ dom Esq

&

Atrib(var3) = atr2 ∈ Esq(c6).Atributos

==> var3.Atributo ∈ ATRIBUTOS

y también se cumple que

var3.Rol ∈ Esq(c5).Roles

Por lo tanto se satisface I33 para var3. Pruebe que también se satisface para var4.

Observe que c8=Clase(var3) debe pertenecer a Esq(c5).Agreg-Rol(var3.Rol) y que c9=Clase(var4) debe pertenecer a Esq(c5).Agreg-Rol(var4.Rol) (este es un requisito necesario para la retribución y será especificado más adelante).

Para auxiliarnos en la especificación de un invariante que impida ciclos dependencia en las definiciones de atributos retribuidos, definiremos una relación, que llamaremos _depende-Ret_ , completamente análoga a la relación definida para atributos heredados.

El selector _depende-Ret_ relacionará un atributo

retribuido de una clase c con un atributo de otra clase del cual depende. En otras palabras, una pareja a , a de tipo ATRIB, estará en la relación _depende-Ret_ si el atributo al que se refiere a (el componente a .Atr) es un atributo retribuido , y si alguna de las variables de la expresión que lo define se refiere al atributo de a (a .Atr) y a su misma clase (a .Cla)

A continuación presentamos la especificación formal del selector _depende-Ret_.

```
=====================================================
|
|     BDM
|
|     _depende-Ret_ : ATRIBUTOS <---> ATRIBUTOS
|
|-------------------
|
|     a  depende-Ret a  <==>
|      i              j
|
|
|     ( a .Atr : Esq (a .Cla).Atr-Ret )
|           i              i
|      &
|
|     ( ∃ v : Variables (Esq(a .Cla).Def-AR(a .Atr)
|                                  i                 i
|               a .Cla = Clase(v)
|                j
|                              &
|
|               a .Atr = Atrib(v)   )
|                j
|_____________________________________________________
```

Ejemplo.

Recuerde que en Esq(c5) tenemos los componentes.

Def-AR = {atr7 !--> var3 + var4}

Atr-Ret = {atr7}

donde

Clase(var3) = c8

Atrib(var3) = atr2

var3.Rol = r1

Clase(var4) = c9

Atrib(var4) = atr1

var4.Rol = r1

Entonces, si a_i está definido por los componentes

$a_i.Cla = c5$, $a_i.Atr = atr7$

y a_j está definido por los componentes

$a_j.Cla = c8$, $a_j.Atr = atr2$

podemos comprobar que

a_i depende-Ret a_j

ya que

atr7 ∈ Esq(c5).Atr-Ret

∧

(∃ var3: Variables(Esq(c5).Def-AR(atr7)) .

$a_j.Cla = c8 = Clase(var3)$

∧

$a_j.Atr = atr2 = Atrib(var3)$)

De la misma forma puede comprobarse que si $a_j.Cla = c9$ y si $a_j.Atr = atr1$ entonces también a_i depende-Ret a_j .

Redefiniremos ahora BDM para incluir el invariante que garantizará que los atributos retribuidos no serán dependientes cíclicamente evitando la indeterminación en sus valores.

BDM

BDM	
$\_depende\text{-}Ret^{+}\_$ Π id ATRIBUTOS = $\{\}$	I34

I34: Atributos retribuidos bien definidos.

Ejemplo.

Supongamos que EBDM tiene definido únicamente a las clases c5, c8 y c9. En este caso se cumple I34. La comprobación se hace en forma similar a otros ejemplos ya vistos.

Hemos visto en los ejemplos que la retribución a una clase se hace desde sus objetos agregados. Este requerimiento será especificado en la siguiente redefinición de BDM, donde se establece que cada variable v que aparece en la definición de atributos retribuidos de clase c, solo puede tener como componente Clase(v) a alguno de los agregados de c bajo el rol v.Rol, es decir, Clase(v) debe pertenecer a Esq(c).Agreg-Rol(v.Rol). Esto implica que la retribución es a un nivel, pero, como en el caso de los atributos heredados, existe la posibilidad de establecer

las cadenas de retribución que sean necesarias.

EBDM

EBDM
∀ c : dom Esq ; exp : rng Esq(c).Def-AR;
v: Variables (exp) .
Clase(v) ∈ Esq(c).Agreg-Rol(v.Rol) I35

I35: Las variables que están en expresiones que definen atributos retribuidos hacen referencia a clases de objetos agregados a través de v.Rol (Retribución a un nivel).

Ejemplo.

Suponga que EBDM tiene definidas las clases c5, c8 y c9, entonces se cumple I35 ya que

Variables(Esq(c5).Def-AR(atr7)) = {var3, var4}

Para var3:

Clase(var3) = c8 ∈ Esq(c5).Agreg-Rol(var3.Rol)=
Esq(c5).Agreg-Rol(r1)={c8,c9}

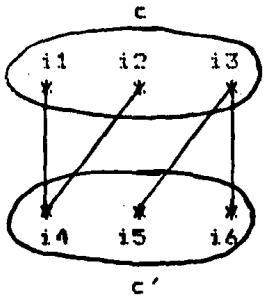
Y para var4:

Clase(var4) = c9 ∈ Esq(c5).Agreg-Rol(var4.Rol)=
Esq(c5).Agreg-Rol(r1)={c8,c9}

V.1.4.2 CONSISTENCIA DE INSTANCIAS.

Un factor importante para la consistencia de instancias cuando existen atributos retribuidos es la ausencia de ambigüedad en la evaluación de dichos atributos. Para la comprensión del problema veremos un ejemplo.

Supongamos que tenemos dos clases con instancias agregadas en la siguiente forma (para el mismo rol de agregación r).



Si la clase c tiene atributos retribuidos que dependen de c', entonces, la evaluación de los atributos retribuidos de i1 e i2 no significará problema alguno; pero en cambio, para la evaluación de los atributos retribuidos de i3 habrá ambigüedad ya que no podrá determinarse quien retribuye, si i5 o i6, ya que ambas podrían hacerlo.

Para evitar esto, es necesario que la asociación entre instancias sea una función (es decir, que el Mapeo sea una función). En la gráfica anterior, el Mapeo no es una función ya que i3 está asociado con i5 e i6 al mismo tiempo, lo cual es imposible en una función.

El invariante que garantiza la ausencia de ambigüedad

en instanciación de atributos retribuidos establece que para toda clase c con atributos retribuidos, y para toda variable v que sea variable de alguna de las expresiones que los definen, debe cumplirse que el Mapeo aplicado a la terna c, v.Rol y Clase(v) debe ser una función.

Introducimos este invariante en la siguiente redefinición de RDM.

RDM

RDM	
∀ c : dom Esq ; exp : rng Esq(c).Def-AR; v: Variables (exp) . Mapeo(c, v.Rol, Clase(v)) ∈	ID-INST--//-->ID-INST I36

I36: Ausencia de ambigüedad en instanciación de atributos retribuidos (Mapeo debe ser función).

Ejemplo.

Supongamos que tenemos una RDM cuyas clases son c y c' definidas por

Esq(c).Def-AR = {atr1 !--> var1}

Esq(c).Agreg-Rol = { r !--> {c'} }

Esq(c').Atributos = {atr1'}

donde

Clase(var1) = c'

.Atrib(var1) = atri'

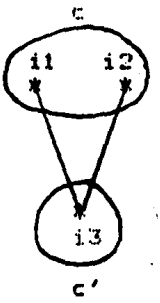
var1.Rol = r

Si

Mapeo(c.var1.Rol.Clase(var1)) = Mapeo (c,r,c')

= {i1 !--> i3,i2 !--> i3}

con i1, i2, i3 pertenecientes a ID-INST, entonces la gráfica es

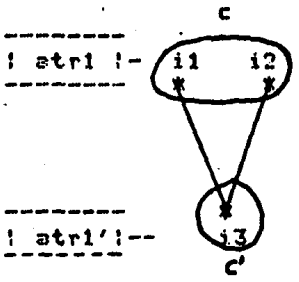


y se cumple que

Mapeo(c,var1.Rol,Clase(var1))∈ID-INST --//-->ID-INST

y por lo tanto se satisface I36.

Para el estudio de la evaluación de atributos retribuidos tomaremos el ejemplo anterior cuya gráfica es



El valor del atributo retribuido atr1 de la instancia i1 de c está dado por el selector Valor-Inst(c,i1,atr1) que será igual a

Valor-Inst(c,i1,atr1)=Evalúa(Ambiente)(exp-AR)
=Evalúa(Ambiente)(var1)

Donde exp-AR = Esq(c).Def-AR(atr1) = var1

El Ambiente en el contexto de atributos retribuidos estará definido por

Ambiente: VAR ----> VALOR

dom Ambiente = Variables(exp-AR)

En nuestro caso

dom Ambiente = var1

Para saber qué valor asocia la función Ambiente a var1, aplicamos la función Valor-Inst en la forma siguiente.

$$Ambiente(var1)=Valor\text{-}Inst(Clase(var1),id^{*}(var1),Atrib(var1))$$

donde $id^{*}(var1)$ es el identificador al que está asociada la instancia i1 (en nuestro ejemplo la instancia asociada es i3). Formalmente

$$id^{*}(var1) = Mapeo(c,var1.Rol,Clase(v))(i1)$$
$$= Mapeo(c,r,c')(i1)$$
$$= i3$$

Note que si Mapeo no fuera una función (como lo garantiza el invariante de ausencia de ambigüedad en instanciación de atributos retribuidos (I36)), entonces $id^{*}(var1)$ no estaría únicamente definido.

El valor del atributo retribuido atr1, de la instancia i2 de c se calcula en la misma forma, el lector puede comprobar que será igual al valor de atr1 para la instancia i1.

En la siguiente redefinición de BDM se introduce el invariante que formaliza y generaliza la evaluación de atributos retribuidos

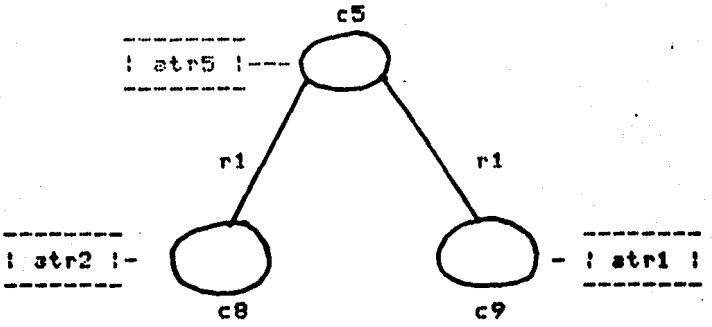
```
BDM_____________________________________________________________
|
|    BDM
|  ____________
|
|    V (c, id, a) : dom Valor-Inst  .
|    (  a ∈ Esq(c).Atr-Ret ) ===>
|      [ V v: Variables(exp-AR) ==>
|                          *
|         (Clase(v), id (v), Atrib(v)) ∈ dom Valor-Inst
|      ]
|      [ Valor-Inst(c,id,a) = EvalÚa(Ambiente)(exp-AR)   I37
|        Donde
|        Ambiente: VAR --//--> VALOR
|        dom Ambiente = Variables(exp-AR)
|        V v: dom Ambiente
|               Ambiente(v) =
|                                    *
|               Valor-Inst(Clase(v), id (v), Atrib(v)) ]
|    Donde
|    exp-AR = Esq(c).Def-AR(a)
|
```

```
*
id (v) = Mapeo(c, v.Rol, Clase(v))(id),
```

I37: El valor de un atributo retribuido de una instancia de c está dado por la evaluación que lo define en el ambiente creado (Evaluación de atributos retribuidos).

Ejemplo.

Tomaremos como ejemplo una BDM cuyo esquema tiene definidas a las clases c5, c8 y c9. Recuerde que la gráfica de esas clases es



Nos interesa también recordar que

Esq(c5).AR = {atr7 !--> ENTERO}

Esq(c5).Def-AR = {atr5 !--> var3 + var4}

Esq(c5).Agreg-Rol = { r1 !--> {c8,c9} }

Donde

Clase(var3) = c8

Atrib(var3) = atr2

var3.Rol = r1

Clase(var4) = c9

Atrib(var4) = atr1

var4.Rol = r1

y donde

Esq(c8).AP = ⟨atr2 !--> ENTERO⟩

Esq(c9).AP = ⟨atr1 !--> ENTERO⟩

Esq(c7).Agreg-Rol = { r2 !--> ⟨c5⟩ }

Supongamos que

Inst(c5).IC = { id7 !--> ⟨atr7 !--> 6⟩ ,
id8 !--> ⟨atr7 !--> 6⟩ }

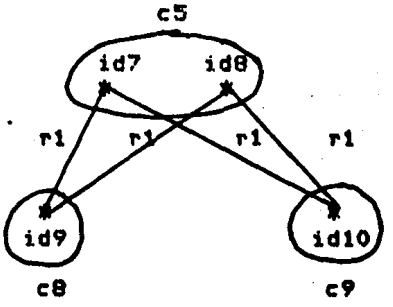
Inst(c8).IC = { id9 !--> ⟨atr2 !--> 5⟩ }

Inst(c9).IC = { id10 !--> ⟨atr1 !--> 1⟩ }

Mapeo(c5,r1,c8) = { id7 !--> id9, id8 !--> id9 }

Mapeo(c5,r1,c9) = { id7 !--> id10, id8 !--> id10 }

La gráfica de instancias sería



Verificaremos que se cumple I37 para (c5,id7,atr7).

(c5,id7,atr7)! dom Valor-Inst

atr7 ∈ Esq(c5).Atr-Her

exp-AR = Esq(c5).Dof-AR(atr7) = var3 + var4

dom Ambiente = Variables(exp-AR)

= Variables(var3 + var4)

= {var3, var4}

∀ v ∈ dom Ambiente .

$$Ambiente(v) = Valor\text{-}Inst(Clase(v), id^{*}(v), Atrib(v))$$

como

$$id^{*}(var3) = Mapeo\ (c5, var3.Rol, Clase(var3))(id7)$$

= Mapeo (c5, r1, c8)(id7)

= id9

y

$$id^{*}(var4) = Mapeo\ (c5, var4.Rol, Clase(var4))(id7)$$

= Mapeo (c5, r1, c9)(id7)

= id10

y además

$$(Clase(var3), id^{*}(var3), Atrib(var3)) = (c8, id9, atr2) \in dom\ Valor\text{-}Inst$$

$$(Clase(var4), id^{*}(var4), Atrib(var4)) = (c9, id10, atr1) \in dom\ Valor\text{-}Inst$$

entonces

Ambiente(var3) = Valor-Inst(c8,id9,atr2) = 5

Ambiente(var4) = Valor-Inst(c9,id10,atr1) = 1

por lo tanto

Valor-Inst(c5,id7,atr7) = Evalúa(Ambiente)(exp-AR)

= Evalúa(Ambiente)(var3+var4)

= 5 + 1 = 6

Lo mismo puede verificarse como para (c5,id8,atr7)

con lo cual se satisface I37. (58)

V.2 DEFINICION DE CONDICIONES SOBRE ROLES.

En el modelo básico hemos especificado ya formalmente el concepto de roles. En esta sección introduciremos en el modelo la posibilidad de restringir las agregaciones de objetos, lo que será posible al definir condiciones sobre los roles de agregación.

Hemos visto ya la importancia que tiene para las capacidades expresivas del modelo el mecanismo de condicionamiento de roles. Una condición sobre un rol será definida por medio de una fórmula, que al evaluarse producirá solo dos posibles valores: VERDADERO o FALSO. En el primer caso, las instancias serían asociadas ; en el segundo, existiría una restricción a la agregación y , por lo tanto, las instancias no serían agregadas o asociadas por el rol.

Las fórmulas que definen a las condiciones sobre los roles deben ser expresiones bien formadas de un lenguaje adecuado a nuestros propósitos. Dicho lenguaje será una extensión del lenguaje de expresiones utilizado para definir atributos. En la siguiente sección veremos algunas de sus características.

V.2.1 EXTENSION DEL LENGUAJE DE EXPRESIONES.

La definición de un lenguaje para condicionamiento de roles puede variar mucho , según las capacidades expresivas que se deseen. Aquí señalaremos las características básicas que debe tener el lenguaje, para que el lector tenga los conceptos necesarios para la comprensión de las siguientes secciones.

El lenguaje necesario para la definición de condicionamiento de roles puede ser una extensión del lenguaje de expresiones usado para definir atributos ya que este lenguaje es suficiente para denotar valores de atributos. Es decir, el significado asociado a cada expresión del lenguaje, dado un medio ambiente específico, es un VALOR. Si extendemos el lenguaje de expresiones, incluyendo predicados cuyo significado en un ambiente sea un BOLEANO definiendo

BOLEANO ≡ ⟨FALSO, VERDADERO⟩

entonces podremos definir un lenguaje de expresiones booleanas que nos permita convinar predicados con los operadores lógicos que sean definidos en la extensión (la especificación formal de la extensión del lenguaje puede consultarse en el Apéndice II).

La extensión del lenguaje de expresiones es un lenguaje de expresiones booleanas al que llamaremos lenguaje de condicionamiento de roles, consistirá de un conjunto de expresiones boleanas bien formadas parametrizadas por el conjunto de nombres de variables permitidos, al que llamaremos EXPBOLBF [NOM-VAR] . Así, existe la posibilidad

de generar una familia de lenguajes distintos dependiendo de NOM-VAR.

Cada lenguaje de condicionamiento de roles tendrá en en común los siguientes elementos:

- Mismo conjunto de predicados. Por ejmplo:
  <, >, =, <>, etc.
- Mismo conjunto de operaciones lógicas. Por ejemplo:
  ==>, &, v, etc.
- Dada una expresión de tipo EXPBOLBF, la función
  Variables aplicada a ella, le asociará el conjunto
  de nombres de variables de dicha expresión.

  $$\text{Variables}_{expb} : EXPBOLBF \dashrightarrow F\ NOM\text{-}VAR$$

  Ejemplo. Suponiendo que x, y son nombres de variables y que " x < y " es una EXPBOLBF entonces

  $$\text{Variables}_{expb}(x > y) = \{x, y\}$$

- Dado un conjunto de nombres de variables, la función Ambiente asociará a cada nombre de variable el VALOR que le corresponda.

  $$Ambiente \equiv NOM\text{-}VAR \dashrightarrow VALOR$$

- La evaluación de una expresión boleana bien formada se obtiene al aplicar la función Evalúa, suponiendo un ambiente determinado.

  $$Evalúa_{expb} : Ambiente \longrightarrow EXPBOLBF \dashrightarrow BOLEANO$$

  El resultado de evaluar una expresión boleana bien for-

mada en un medio ambiente es VERDADERO si la relación se satisface, y FALSO en caso contrario.

Ejemplo. Si " (x < y) & (y = z) " es una EXPBOLBF donde x, y, z son nombres de variables , donde < es un predicado y donde el ambiente esta dado por

Ambiente = { x !--> 1, y !--> 2, z !--> 3 }

entonces

Evalúa (Ambiente)((x<y) & (y=z)) = FALSO
expb

En la siguiente sección veremos como se utiliza un lenguaje de expresiones boleanas en la definición de condicionamiento de roles.

V.2.2 CONDICIONAMIENTO SOBRE ATRIBUTOS.

El condicionamiento de roles sobre atributos es una capacidad del modelo mediante la cual se podrán restringir agregaciones que, aunque estén bien definidas, no satisfagan ciertas condiciones sobre los valores de los atributos de instancias particulares de los objetos a asociar.

Para definir el condicionamiento de roles sobre atributos, incluiremos un componente más en DEF-CLASE, al cual llamaremos Cond/at.

La función Cond/at asocia a cada agregado (a través de un rol especifico) la expresión boleana bien formada

que define la condición .

Los nombres de variables permitidos, que parametrizarán a EXPBOLBF, serán del tipo ATRIB ya definido . Al lenguaje de expresiones boleanas bien formadas parametrizado por ATRIB le llamaremos lenguaje de condicionamiento sobre atributos. Las expresiones de este lenguaje serán llamadas EXPB/AT, es decir, se cumple que

EXPB/AT ≡ EXPBOLBF [ATRIB]

Para que las condiciones estén bien definidas deben establecerse sobre roles definidos y sobre agregados definidos sobre esos roles.

En la siguiente redefinición de DEF-CLASE se introduce el componente Cond/at y los dos invariantes necesarios para que las condiciones estén bien definidas. Se incluye antes la definición del conjunto de expresiones de condiciones sobre atributos.

EXPB/AT ≡ EXPBOLBF [ATRIB];

DEF-CLASE

DEF-CLASE

Cond/at: NOM-ROL -//-> (NOM-CLASE -//-> EXPB/AT)

dom Cond/at C Roles I38

∀ r: dom Cond/at .

dom Cond/at(r) C Agreg-Rol(r) I39

I38: Condición establecida sobre roles definidos.

I3Q: Condición establecida sobre agregado adecuado.

Ejemplo.

Retomemos un ejemplo ya visto en la sección que formaliza el concepto de rol (IV.1.3). Sea c4 definido de la forma siguiente.

Esq(c4).AP = ⟨atr1 :--> ENTERO, atr2 :--> ENTERO ⟩

Esq(c4).Atributos = ⟨atr1, atr2⟩

Esq(c4).Agreg-Rol = ⟨ r1 :--> ⟨c1,c2⟩ ,
r2 :--> ⟨c2,c3⟩ ⟩

Esq(c4).Roles = ⟨ r1, r2 ⟩

Esq(c4).Agregados = ⟨ c1, c2, c3 ⟩

donde atr1, atr2 son nombres de atributos; r1, r2 son nombres de roles y donde c1, c2 y c3 están definidos por

Esq(c1).AP = ⟨atr1 :--> ENTERO⟩

Esq(c1).Atributo = ⟨atr1⟩

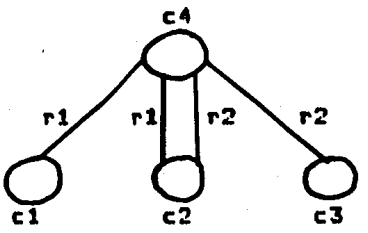
Esq(c2).AP = ⟨atr2 :--> ENTERO⟩

Esq(c2).Atributo = ⟨atr2⟩

Esq(c3).AP = ⟨atr3 :--> ENTERO⟩

Esq(c3).Atributo = ⟨atr3⟩

La gráfica que representa la situación es



Si las condiciones sobre atributos están definidas por

$$Esq(c4).Cond/at = \{r1 \mapsto (c1 \mapsto ((var1 < var3) \vee (var2 > var3)),\ r1 \mapsto (c2 \mapsto (var1 = var4),\ r2 \mapsto (c3 \mapsto (var2 \neq var5)\}$$

donde var1, var2, var3, var4 y var5 son de tipo ATRIB y están definidas por

$var1.Cla = c4,\ var1.Atr = atr1$

$var2.Cla = c4,\ var2.Atr = atr2$

$var3.Cla = c1,\ var3.Atr = atr1$

$var4.Cla = c2,\ var4.Atr = atr2$

$var5.Cla = c3,\ var5.Atr = atr3$

Entonces, se cumple el invariante I38, ya que

$$dom\ Esq(c4).Cond/at = \{r1, r2\} \subset Esq(c4).Roles$$

y se cumple I39 ya que

$$dom\ Cond/at(r1) = \{c1, c2\} \subset Esq(c4).Agreg\text{-}Rol(r1) = \{c1, c2\}$$

$$dom\ Cond/at(r2) = \{c3\} \subset Esq(c4).Agreg\text{-}Rol(r2) = \{c2, c3\}$$

Para facilitar la notación de variables tipo ATRIB haremos la siguiente convensión.

c.a = μ { v:ATRIB/ v.Cla = c & v.Atr = a }

Nota Técnica.- El selector μ obtiene un elemento del conjunto.

Ejemplo.

Las variables var1, var2, var3, var4 y var5 se denotarán de la siguiente forma, de acuerdo a la convensión

var1 = c4.atr1

var2 = c4.atr2

var3 = c1.atr1

var4 = c2.atr2

var5 = c3.atr3

Puesto que en varias ocasiones será necesario hacer referencia a una expresión que defina una condición sobre atributos, definiremos el selector auxiliar Def-Cond/at.

La función Def-Cond/at asociará a una clase c_i, un rol de ella r y un agregado bajo ese rol c_j, la expresión que define su condición sobre atributos.

BDM

Def-Cond/at:

NOM-CLASE X NOM-ROL X NOM-CLASE ---> EXPR/AT

$$\text{Def-Cond/at} = \{ (c_i, r, c_j) \dashrightarrow \text{Esq}(c_i).\text{Cond/at}(r)(c_j) / (c_i \in \text{dom Esq}, r: \text{dom Esq}(c_i).\text{Cond/at}) \}$$

(c ∈ dom rng Esq(c).Cond/at(r))}
 j i

Ejemplo.

En nuestro caso de estudio, las definiciones de condiciones se obtienen aplicando el selector Def-Cond/at de la siguiente forma.

Def-Cond/at(c4,r1,c1) = (c4.atr1 < c1.atr1) ∨
(c4.atr2 > c1.atr1)

Def-Cond/at(c4,r1,c2) = (c4.atr1 = c2.atr2)

Def-Cond/at(c4,r2,c3) = (c4.atr2 <> c3.atr3)

Las expresiones que definen condiciones sobre atributos, solo podrán tener variables que se refieran ya sea a la clase dónde se define la condición y atributos de ella, o ya sea al agregado para el que se define la condición y atributos de él. No se establecerá ninguna limitación en cuanto al tipo de atributos (es decir, se podán condicionar atributos primitivos, generados, heredados y retribuidos.

En el siguiente esquema se redefine EBDM, incluyendo el invariante necesario.

EBDM

EBDM

$$\forall (c_i, r, c_j) : \text{dom Def-Cond/at};\ v \in \text{Variables}_{expb}(\text{Def-Cond/at}(c_i, r, c_j)) .$$

$$(\text{Clase}(v) \,\&\, \text{Atrib}(v) \in \text{Esq}(c_i).\text{Atributos}_v)$$

$$(\text{Clase}(v) \,\&\, \text{Atrib}(v) \in \text{Esq}(c_j).\text{Atributos}) \quad \text{I40}$$

I40: Las variables de una expresión que define una condición sobre atributos sólo pueden referirse a atributos de la clase definida o de su agregado.

Ejemplo.

El invariante I40 se satisface en nuestro caso de estudio ya que

(c4,r1,c1): dom Def-Cond/at

$\text{Variables}_{expb}(\text{Def-Cond/at}(c4,r1,c1)) = \{c4.atr1, c4.atr2, c1.atr1\}$

Para c4.atr1 se cumple que

Clase(c4.Atr1) = c4

&

Atrib(c4.atr1) = atr1 ∈ Esq(c4).Atributos ={atr1,atr2}

Para c4.atr2

Clase(c4.atr1) = c4

&

Atrib(c4.atr2) = atr2 ∈ Esq(c4).Atributos ={atr1,atr2}

Y para c1.atr1

Clase(c1.Atr1) = c1

Atrib(c1.atr1) = atr1 ∈ Esq(c1).Atributos = {atr1}

De forma análoga se comprueba que (c4,r1,c2) y (c4,r2,c3) satisfacen el invariante.

Para garantizar que las instancias asociadas a través de roles solo sean las que cumplen las condiciones sobre atributos es necesario evaluar la expresión que define la condición en el Ambiene adecuado.

Es decir, siempre que dos instancias (i_1, i_2) estén en el Mapeo (lo que implica que están asociadas) se deberá cumplir que la evaluación de condición sobre atributos (si la hay) sea verdadera.

El Ambiente necesario para la evaluación tiene la siguiente signatura en el contexto del lenguaje de condiciones sobre atributos.

Ambiente: ATRIB ----> VALOR

El dominio de Ambiente son todas las variables que se encuentran en la definición de la condición. El valor que se asociará a cada variable estará determinado por la instancia i especifica a que se haga referencia por lo que se denotará como Ambiente-Inst[i](v). Así, si v = c.a, entonces

Ambiente-Inst[i](v) = Valor-Inst(c,i,a)

La siguiente redefinición de BDM incluye estos aspectos, introduciendo un nuevo invariante.

BDM

BDM

∀ (c ,r , c) : dom Mapeo .
  i j
(c ,r, c) ∈ dom Def-Cond/at ==>
 i j

[∀ v:Variables (Def-Cond/at(c ,r,c) .
  expb i j
v ∈ dom Ambiente

&

[∀(i ,i) ∈ Mapeo(c ,r , c) .
  1 2 i j

Evalúa (Ambiente[i ,i])(Def-Cond/at(c , r, c))
  expb 1 2 i j

= VERDADERO I41

Donde

Ambiente[i ,i] ≡ Ambiente-Inst[i] ∪ Ambiente-Inst[i]
  i j i j

Ambiente-Inst[i] ≡ {c.a |--> val/ (c,i,a) ∈ Valor-Inst &

val = Valor-Inst(c,i,a)

I41: Restricción del mapeo a las instancias que satisfagan la Cond/at.

Nota Técnica.- Ambiente se definió a través de una equivalencia sintáctica con parámetros. Ambiente[i ,i] se obtiene al sustituir sistemáticamente i por i y i por i en todo el lado derecho de la definición Ambiente[i ,i]. Un proceso similar se sigue con Ambiente-Inst[i] que se obtiene al sustituir cada ocurrencia de i por i . Analogamente se obtiene Ambiente-Inst[i].

Ejemplo.

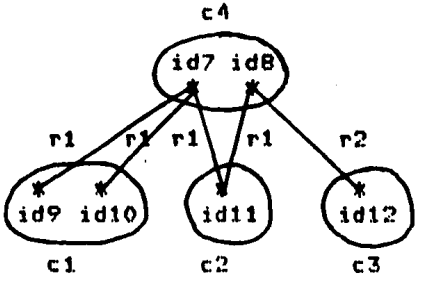
La función Mapeo aplicada a nuestro caso de estudio había sido definida por

Mapeo (c4, r1, c1) = {id7 !--> id9, id7 !--> id10}

Mapeo (c4, r1, c2) = {id7 !--> id11,id8 !--> id11}

Mapeo (c4, r2, c3) = {id8 !--> id12}

La siguiente gráfica representa la situación si hacemos caso omiso de las condiciones sobre atributos.



Supongamos que las instancias tienen los siguientes valores

Inst(c4).IC = { id7 !--> {atr1 !--> 2,atr2 !-->1},
id8 !--> {atr1 !--> 4,atr2 !-->0} }

Inst(c1).IC = { id9 !--> {atr1 !--> 5} ,
id10 !--> {atr1 !--> 1} }

Inst(c2).IC = { id11 !--> {atr2 !--> 4} }

Inst(c3).IC = { id12 !--> {atr3 !--> 1} }

Al evaluar las condiciones de atributos veremos cuales asociaciones no serán permitidas. Analicemos la primera pareja (id7,id9) perteneciente a Mapeo(c4,r1,c1).

El Ambiente asociado a ella tendrá como dominio a

dom Ambiente[id7,id9] = Variables(Def-Cond/at(c4,r1,c1))

= Variables((c4.atr1 < c1.atr1) v

(c4.atr2 > c1.atr1))

= {c4.atr1, c4.atr2, c1.atr1}

La definición de Ambiente indica que

Ambiente-Inst[id7] = {c4.atr1 |-->2, c4.atr2 |--> 1}

Ambiente-Inst[id9] = {c1.atr1 |--> 5}

Ambiente[id7,id9] = {c4.atr1 |--> 2,c4.atr2 |--> 1,

c1.atr1 |--> 5 }

Obtenido ya el Ambiente, podemos evaluar

Evalúa (Ambiente[id7,id9])((c4.atr1 < c1.atr1) v
expb

(c4.atr2 > c1.atr1))=

(2 < 5) v (1 > 5) = VERDADERO

Por lo tanto, la asociación sí está permitida, es decir, (id7,id9) ∈ Mapeo(c4,r1,c1).

Analicemos ahora la pareja (id7,id10) y su Ambiente asociado

dom Ambiente[id7,id10] = {c4.atr1, c4.atr2, c1.atr1}

Ambiente-Inst[id10] = {c1.atr1 |--> 1}

Ambiente[id7,id10] = {c4.atr1 |--> 2, c4.atr2 |--> 1,

c1.atr1 |--> 1}

Entonces, la evaluación es la siguiente

Evalúa (Ambiente[id7,id10])((c4.atr1 < c1.atr1) v
expb

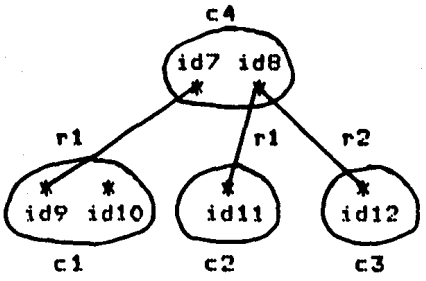
(c4.atr2 > c1.atr1))=

(2 < 1) v (1 > 1) = FALSO

Por lo tanto, la asociación no se permitirá y (id7,id10) no debe pertenecer a Mapeo(c4,r1,c1).

Pueden analizarse en forma similar las demás parejas: (id7,id11), (id8,id11), (id8,id12) y comprobar que el condicionamiento sobre atributos elimina varias asociaciones de Mapeo.

La gráfica que representa la situación considerando condicionamiento sobre atributos es la siguiente.



Con este ejemplo terminamos la especificación formal del modelo extendido. En el siguiente capítulo trataremos las operaciones en BDM y su formalización.

CAPITULO VI.

FORMALIZACION DE OPERACIONES.

En este capitulo se formalizarán algunas operaciones básicas sobre instancias de una base de datos molecular, como son inserción, desconexión, borrado, conexión y algunas variantes de ellas.

Por lo general, se presentará en primer término la operación en su variante más simple , para después exponer las operaciones más elaboradas utilizando operaciones previamente definidas, de forma tal que se simplifique lo más posible la especificación.

Una operación puede ser exitosa o no exitosa. LLamaremos operación total a la operación que haya sido especificada para todos los casos posibles, es decir, éxito o fracaso de la operación.

Para cada operación se formalizará primero la operación exitosa, en seguida se especifican los casos posibles de fracaso o no éxito de la operación y por último se esespecifica la operación total.

En ocasiones, cuando se traten algunas variantes de operaciones definidas en términos de operaciones totales más elementales, el éxito o el fracaso dependerá de dichas operaciones más elementales. En estos casos solo

será necesario especificar la operación total.

VI.1 ESTADO INICIAL DE BDM.

En la definición de las operaciones supondremos que que en el estado inicial de BDM su IBDM no tiene ninguna instancia para ninguna clase, que la instanciación de roles solo puede hacerse sobre roles definidos y que no existe ningún rol instanciado.

En el siguiente esquema, al cual llamamos I-BDM, especificamos formalmente el estado inicial de BDM.

I-BDM

BDM

∀ c : dom Inst .

Inst(c).IC = {}

Inst(c).Instancias = {}

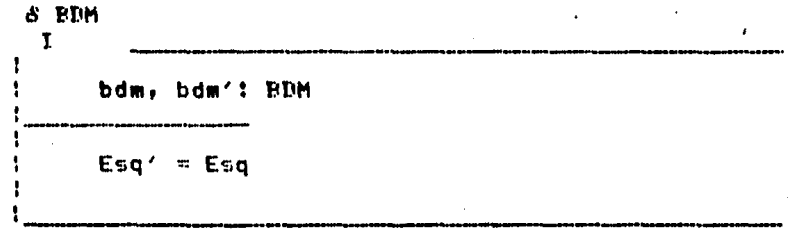
dom Esq(c).Roles = dom.Inst(c).R

rng Inst(c).R = { {} }

En los siguientes esquemas, se especifican formalmente los efectos que ocasionan diversos tipos de operaciones sobre BDM.

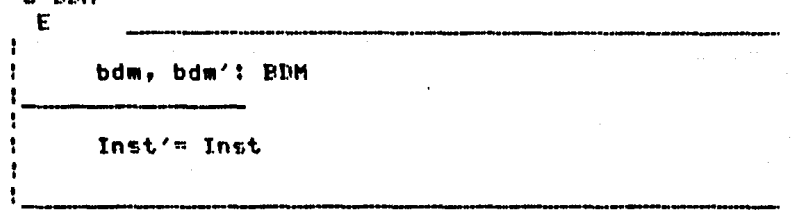
Las operaciones exitosas sobre instancias ocasionan

cambio en IBDM pero EBDM queda inalterado. Llamamos δ BDM I al esquema que especifica esta situación, indicando con δ_I que es posible que haya cambio en IBDM.



Nota Técnica.- Las variables sin apóstrofo indican el estado anterior a la operación. Las variables con apóstrofo indican el estado posterior a la operación.

Las operaciones exitosas sobre el Esquema (como por ejemplo la definición de una clase) pueden ocasionar cambio en EBDM pero no en IBDM. LLamamos δ_E BDM al esquema correspondiente, indicando con δ_E que es posible que haya cambio en EBDM.



Las operaciones de consulta de información o las operaciones no exitosas sobre el esquema o instancias dejan inalterado EBDM e IBDM. Llamamos ΞBDM al siguiente esquema, para indicar que no habrá cambio.

```
≡BDM
|
|     bdm, bdm': BDM
|_____________
|
|     Esq' = Esq
|
|     Inst'= Inst
|
|_____________________________________________________________
```

VI.2 INSERCION.

En general, la operación de inserción se refiere al hecho de insertar una nueva instancia de una clase de objeto previamente definida. Al insertar una instancia, el sistema debe asignarle un identificador único, que no haya sido asignado aún a ninguna instancia.

En el siguiente esquema redefiniremos a BDM, agregando un componente derivado, al que llamaremos Nuevos-Id, y un predicado que establecerá que el conjunto Nuevos- Id será el conjunto de identificadores de instanicia que aún no han sido asignados a ninguna instancia de ninguna clase en BDM.

```
BDM
|
|     BDM
|
|     Nuevos-Id : F ID-INST
|_____________
|
|     Nuevos-Id = { id: ID-INST/ V c:dom Inst,
|
```

id ∉ Inst(c).Instancias)

Cuando se efectúa una inserción, deben proporcionarse dos parámetros de entrada: la clase de la instancia a insertar y la instancia (nombres de atributos asociados a valores).

El siguiente esquema servirá de marco general para las operaciones de inserción, por lo que le hemos llamado MARCO-Inserta, en él se incluyen los siguientes componentes que serán necesarios para toda operación de inserción: las dos variables de entrada ya mencionadas a las que llamaremos c? , i? ; una variable o parámetro de salida al cual llamaremos rep! que indicará si la operación fue exitosa o no.

Antes de presentar el esquema MARCO-Inserta se incluye la definición sintáctica de REPORTE, que es el tipo de la variable rep!. Reporte consistirá de una secuencia de caracteres. Por ejemplo, en el caso de una operación exitosa, rep! será igual a la secuencia de caracteres "OK".

REPORTE ≡ Seq CARACTER

MARCO-Inserta

c?: NOM-CLASE

i?: NOM-ATR --//--> VALOR

rep!: REPORTE

Nota Técnica.- Este esquema auxiliará en la especi-

ficación de la inserción. No tiene predicados, solo declaraciones.

VI.2.1 INSERCION EXITOSA.

A una operación de inserción exitosa le llamaremos INSERTA-Nuevo. El esquema que especifica esta operación tendrá en su parte de declaraciones a MARCO-Inserta. También se incluirá en las declaraciones al esquema δ BDM, lo que inidicará que es posible que haya un cambio en IBDM y que EBDM no será afectado.

Los predicados de todo esquema que especifica una operación exitosa serán de dos tipos:

- Predicados que determinan las características que deben satisfacer los parámetros de entrada para que la operación sea exitosa. A este tipo de predicados se les conoce como precondiciones, por lo que las etiquetaremos con la palabra "PRE".
- Predicados que determinan el valor de los parámetros de salida y que establecen los efectos de la operación, es decir, que definen el estado posterior a la operación exitosa de los componentes con apóstrofo del esquema. A este tipo de predicados se les conoce con el nombre de postcondiciones por lo que las etiquetaremos con la palabra "POS".

Las precondiciones que definiremos en el esquema de INSERTA-Nuevo establecerán que la clase de la instancia

a insertar sea una clase definida en EBDM, y que la instancia a insertar sea consistente con la definición de atributos primitivos de la clase, es decir, que los nombres de atributos primitivos estén definidos y que sus correspondientes valores pertenezcan a los valores del dominio definido.

Note que estas precondiciones para que la operación de inserción sea exitosa no establecen limitantes respecto a un mínimo de atributos de la instancia. Además no condicionan la inserción a casos de instancias que no se encuentren ya en IBDM , es decir , la inserción será exitosa aún en el caso de que haya una o más instancias iguales de esa clase ya insertadas.

Se definirán dos postcondiciones, una reportará el éxito de la operación, la otra establecerá cual será el estado de IBDM después de la operación. Es decir, a Inst --que es un conjunto de nombres de clase asociados a sus correspondientes INST-CLASE-- se le sobre-escribe la pareja que asocia la clase c? a la INST-CLASE nueva, a la que llamaremos I-Nueva, ésta incluirá a la instancia insertada (i?). El operador que utilizaremos para modelar esto será el de sobre-escritura (⊕) (se recomienda revisar en la introducción a Z la definición de esta operación). El resultado de la operación será asignado a Inst'.

El siguiente es el esquema de INSERTA-Nuevo.

INSERTA-Nuevo

δ RDM
I

MARCO-Inserta

---

c?: dom Esq — PRE1

dom i? ⊂ Esq(c?).Atr-Prim — PRE2

∀ a : dom i? .

i?(a) ∈ Val-Dom(Dominio(c?,a)) — PRE3

rep! = "OK" — POS1

Inst'=Inst ⊕ {c? !--> I-Nueva} — POS2

Donde

I-Nueva = Inst(c?)/IC

I-Nueva.IC = Inst(c?).IC ∪ {idN !-->i?}

donde

idN : Nuevos-Id

PRE1: Clase definida.

PRE2: Los nombres de atributos de la instancia corresponden a nombres de atributos primitivos definidos.

PRE3: Los valores de la instancia pertenecen a los dominios correctos.

POS1: Reporte de inserción exitosa.

POS2: Se incorpora una nueva instancia a la clase.

Nota Técnica.- Suponiendo que A es cualquier esquema y X uno de sus componentes, la notación A/X representa a un esquema que es igual al esequema A en todos sus componentes, excepto en X.

VI.2.2 OPERACION TOTAL DE INSERCION.

Para definir la operación total de inserción es necesario considerar los casos en que la inserción no será exitosa. Dichos casos son los siguientes:

- El parámetro de entrada c? no es una clase definida en BDM, es decir, la clase de la instancia a insertar no existe en el dominio de Esq.
- Algunos de los atributos de la instancia a insertar no está definido, es decir, existe al menos un nombre de atributo de la instancia que no pertenece a los atributos primitivos definidos.
- Algún valor de atributo de la instancia a insertar no es consistente con la definición, es decir, dicho valor no pertenece al dominio definido para el atributo correspondiente.

Note que los casos de inserción no exitosa son la negación de las precondiciones de éxito de la operación (esto será así para todas las demás operaciones). Especificaremos un esquema para cada caso de inserción no exitosa, los tres esquemas tendrán las mismas declaraciones, incluirán a MARCO-Inserta y al esquema ≡BDM, que indica que no habrá ningún cambio en EBDM ni en IBDM.

Los predicados de todo esquema que especifica una operación no exitosa son de dos tipos:

- Precondiciones que establecen las condiciones que cumplen los parámetros de entrada cuando la operación es no exitosa.
- Postcondiciones que determinan el valor de los parámetros de salida cuando no tiene éxito la operación.

Puesto que todas las operaciones no exitosas que serán especificadas no causarán ningún cambio sobre BDM, siempre se incluirá como componente en los esquemas de este tipo al esquema ΞBDM lo que implica que no será necesario incluir postcondiciones para efectos de la operación.

Los esquemas que definen operaciones no exitosas de inserción serán llamados CLASE-NoDef, ATR-NoDef y VALOR-Incon .

CLASE-NoDef

ΞBDM

MARCO-Inserta

---

c? ∉ dom Esq PRE4

rep!= "Clase no definida" POS3

PRE4: Clase no definida.

POS3: Reporte de inserción no exitosa.

ATR-NoDef

ΞBDM

MARCO-Inserta

```
    ∃ a : dom i?  .
        a ∉ Esq(c?).Atr-Prim                          PRE5

    rep!= "Atributo no definido"                      POS4
```

PRE5: Atributo no definido.

POS4: Reporte de inserción no exitosa.

```
VALOR-Incon
    ≡BDM
    c?: NOM-CLASE
    i?: NOM-ATR --//--> VALOR
    ∃ a : dom i?  .
        i?(a) ∉ Val-Dom(Dominio(c?,a))                PRE6

    rep!= "Valor inconsistente"                       POS5
```

PRE6: Valor inconsistente.

POS5: Reporte de inserción no exitosa.

La operación total de inserción debe considerar tanto los casos de operación no exitosa como el caso en que la operación sí lo es.

Llamaremos INSERTA a la operación total de inserción. INSERTA será definido como la disyunción de la operación exitosa de inserción y las no exitosas. A continuación se incluye en el modelo la especificación de INSERTA.

INSERTA ≡ INSERTA-Nuevo ∨ CLASE-NoDef ∨

ATR-NoDef ∨ VALOR-Incon

VI.3 DESCONEXION.

En general, la operación de desconexión se refiere al hecho de desconectar instancias de una molécula de instancias de agregados de dicha molécula. Es posible definir diversas variantes de esta operación de acuerdo a necesidades específicas de aplicación, nosotros hemos elegido algunas de esas variantes para mostrar su especificación en los siguientes apartados.

Para auxiliarnos en la formalización, definiremos tres selectores :

- Instancia, es una función que dada una clase y un identificador de instancia de esa clase proporciona el conjunto de nombres de atributos y los respectivos valores correspondientes al identificador.
- Mapeo-agreg, es una función que al aplicarla a una clase y un rol proporciona las clases agregadas a través de ese rol y las parejas de identificadores asociados.
- Iden-Asoc, es una función que dada una clase y una instancia de ella (conjunto de atributos asociados a valores), proporciona el conjunto de identificadores de instancia correspondiente.

Hay que observar que el tercer selector, Iden-Asoc, asocia a una clase y una instancia no sólo un identifica-

dor, ya que es posible que haya en BDM varias instancias iguales pero con distintos identificadores (59).

Los selectores quedarán definidos por el esquema siguiente.

```
BDM
Instancia: NOM-CLASE X ID-INST --//-->
                          (NOM-ATR --//--> VALOR)
Mapeo-Agreg: NOM-CLASE X NOM-ROL --//-->
      (NOM-CLASE -//-> (ID-INST <-->ID-INST))
Iden-Asoc:
     NOM-CLASE X (NOM-ATR -//-> VALOR) -//->
                                    F ID-INST

-----------------
Instancia = { (c, id) !--> Inst(c).IC(id) /
          c: dom Inst, id: Inst(c).Instancias}
Mapeo-Agreg = { (c, r) !--> Inst(c).R(r) /
                c:dom Inst, r: dom Inst(c).R}

V i: (NOM-ATR --//--> VALOR) ;
  c: dom Inst    .
          Iden-Asoc(c,i) = {id: ID-INST/
                            i C Inst(c).IC(id)}
```

VI.3.1 DESCONECTA INSTANCIA DE INSTANCIA.

La operación de desconexión mas simple es la desconexión de una instancia molecular [60] de otra instancia, la cual debe pertenecer a un agregado de la molécula .

Esta es una operación auxiliar para otras operaciones accesibles al usuario aunque ésta no lo es, ya que un parámetro de entrada necesario para poder referirse a una sola instancia especifica del agregado es un identificador de instancia [61] .

Por lo general, para operar sobre instancias asociadas por un rol, deben proporcionarse los siguientes parámetros de entrada:

- La instancia molecular i? .
- La molécula m? a la que pertenece la instancia i?.
- El rol r? .
- El agregado a?, cuya instancia o instancias se asocian con i? a través de r? .

Llamaremos MARCO-Inst-Rol al siguiente esquema que especifica los parámetros necesarios para operar sobre instancias asociadas a través de un rol. Sus componentes serán, además de los parámetros de entrada anteriores, el parámetro de salida rep! que reporta el resultado de la operación y el componente δ BDM indicando que EBDM no será afectado pero que en IBDM podrá haber cambios.

MARCO-Inst-Rol

δ BDM
I
m?: NOM-CLASE

a?: NOM-CLASE

r?: NOM-ROL

i?: NOM-ATR --//--> VALOR

rep!: REPORTE

Antes de especificar la primera operación de desconexión, definiremos otro esquema auxiliar que será también útil para otras operaciones.

Llamaremos PRE-Inst-Rol al esquema que especifica los parámetros básicos para una operación sobre instancias asociadas por un rol y las precondiciones que dichos parámetros deben satisfacer. Las precondiciones establecerán que m? y a? son clases definidas; que r? es un rol de m?; que a? es un agregado de m? a través de r?; que existe por lo menos un identificador, llamado id, que está asociado a la instancia i? de m?, es decir, que existe dicha instancia molecular; y que los atributos de la instancia a desconectar son atributos primitivos .

PRE-Inst-Rol

MARCO-Inst-Rol

id?: ID-INST

---

m? $\in$ dom Inst PRE7

a? $\in$ dom Inst PRE8

```
    r? ∈ Esq(m?).Roles                                    PRE9
    a? ∈ Esq(m?).Agreg-Rol(r?)                            PRE10
    ∃ id: Inst(m?).Instancias  .
         id ∈ Iden-Asoc(m?,i?)                            PRE11
    dom i? C Esq(m?).Atr-Prim                             PRE12
```

PRE7: Molécula definida.

PRE8: Agregado definido.

PRE9: Rol definido.

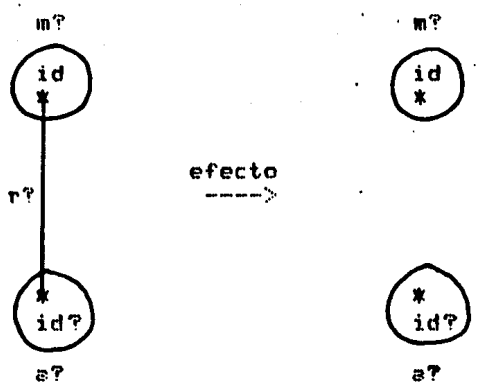
PRE10: Agregado correspondiente a rol definido.

PRE11: Instancia definida.

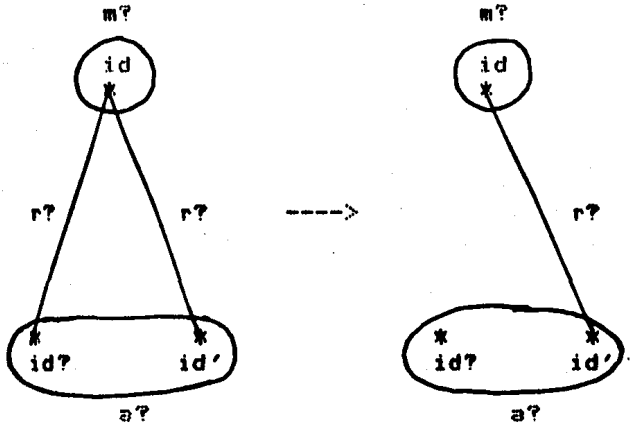
PRE12: Atributos de la instancia primitivos.

Los dos esquemas anteriores nos permitirán especificar ahora, de una forma más concisa, la operación exitosa para desconexión de una instancia de otra, la cual será llamada DESCONECTA-I*. Puesto que esta operación es una operación sobre instancias asociadas por un rol incluiremos al esquema PRE-Inst-Rol, además se definirá un componente más en DESCONECTA-I* llamado id? que será el identificador de la instancia que se desconectará de i?. Se incluirá un predicado, PRE13, que establecerá que id? es un identificador de instancia de la clase a?.

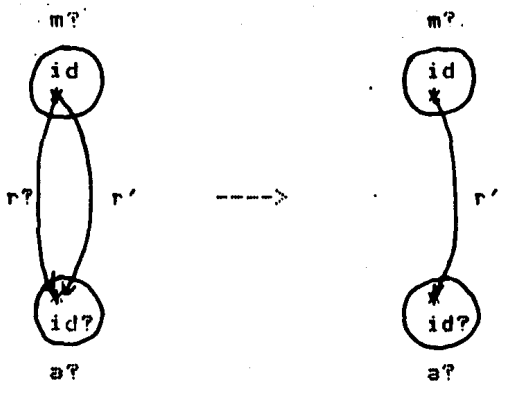
El efecto de la operación será establecido por la postcondición (POS6), dicho efecto se ilustra en la gráfica siguiente.



Si id estuviera asociado a otras instancias de a? el efecto sería como se ilustra enseguida.



Esta operación solo se refiere a la desconexión de instancias asociadas por un rol, en este caso r?, por lo tanto, para casos con más roles, el ejemplo siguiente muestra el efecto.



En el siguiente esquema se especifica la operación DESCONECTA-I*.

DESCONECTA-I*

PRE-Inst-Rol

id?: ID-INST

---

id? ∈ Inst(a?).Instancias PRE13

Inst' = Inst ⊕ {m? |--> I-Nueva} POS6

Donde

I-Nueva = Inst(m?)/R

I-Nueva.R = Inst(m?).R ⊕

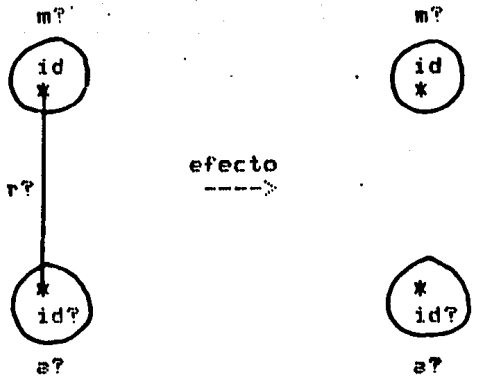
{r |--> (Mapeo-Agreg(m?,r?) ⊕

{a? |--> Mapeo-Nuevo})}

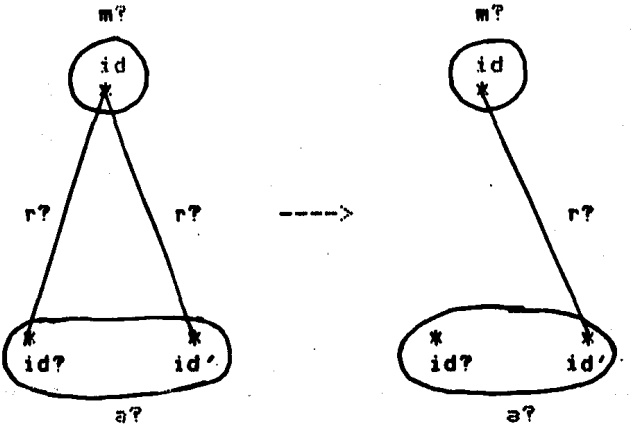
donde

Mapeo-Nuevo = Mapeo(m?, r?, a?) -

{id |--> id? / id ∈ Iden-Asoc(m?,i?)}



Si id estuviera asociado a otras instancias de a? el efecto sería como se ilustra enseguida.



Esta operación solo se refiere a la desconexión de instancias asociadas por un rol, en este caso r?, por lo tanto, para casos con más roles, el ejemplo siguiente muestra el efecto.

PRE13: Identificador perteneciente al agregado.

POS6: Desconexión de la instancia de instancia.

La operación total de desconexión de una instancia molecular de otra instancia será llamada, DESCONECTA-IM*, para especificar esta operación definiremos un esquema para cada caso de operación no exitosa. Los esquemas tendrán las mismas declaraciones que DESCONECTA-I* y como predicado cada esquema tendrá a la negación de la precondición que no se satisface.

MOLECULA-NoDef

MARCO-Inst-Rol	
id?: ID-INST	
m? ∉ dom Inst	PRE14
rep!: "Molécula no definida"	POS 7

AGREGEGADO-NoDef

MARCO-Inst-Rol	
id?: ID-INST	
a? ∉ dom Inst	PRE15
rep!: "Agregado no definido"	POS8

ROL-NoDef

MARCO-Inst-Rol	

```
    id?: ID-INST
-------------------
    r? ∉ Esq(m?).Roles                                  PRE16
    rep!: "Rol no definido"                             POS9
```

AGR-NoRol

```
    MARCO-Inst-Rol
    id?: ID-INST
-------------------
    a? ∉ Esq(m?).Agreg-Rol(r?)                          PRE17
    rep!: "El agregado no corresponde al rol"           POS10
```

INSTANCIA-NoDef

```
    MARCO-Inst-Rol
    id?: ID-INST
-------------------
    ∀ id : Inst(m?).Instancias .
        i? ∉ Instancia(m?,id)                           PRE18
    rep!: "Instancia no definida"                       POS11
```

ATR-NoPrim

```
    MARCO-Inst-Rol
    id?: ID-INST
-------------------
    dom i? ∉ Esq(m?).Atr-Prim                           PRE19
    rep!: "Atributos de instancia no primitivos"        POS12
```

```
ID-NoDef
    MARCO-Inst-Rol
    id?: ID-INST
    id? ∉ Inst(a?).Instancias                          PRE20
    rep!: "Identificador no perteneciente al agregado"POS13
```

A continuación se incluye en el modelo la especificación de la operación total de desconexión de una instancia de otra.

$$DESCONECTA\text{-}IM \equiv DESCONECTA\text{-}I^{*} \vee MOLECULA\text{-}NoDef^{*} \vee AGREGADO\text{-}NoDef \vee ROL\text{-}NoDef \vee AGR\text{-}NoRol \vee INSTANCIA\text{-}NoDef \vee ID\text{-}NoDef \vee ATR\text{-}NoPrim$$

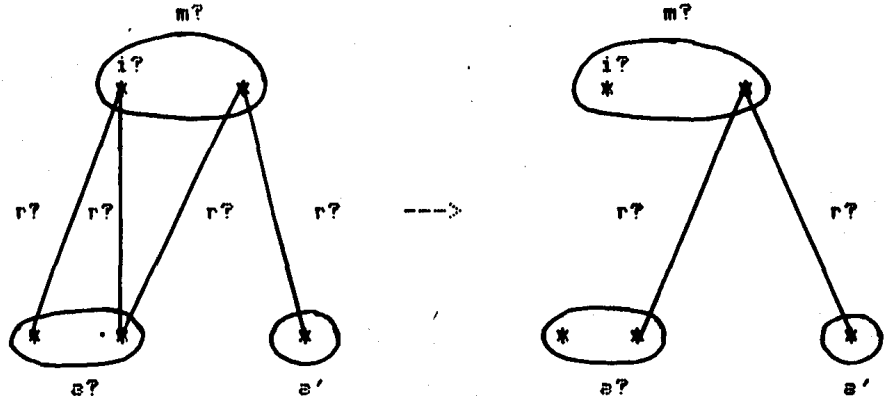
VI.3.2 DESCONECTA INSTANCIA MOLECULAR DE AGREGADO.

La operación de desconexión de una instancia molecular de un agregado será llamada DESCONECTA-IM. El nombre de esta operación, a diferencia de la anteriormente especificada, no está marcado con un asterisco debido a que sí es accesible al usuario, ya que no es necesario dar como parámetro de entrada ningún identificador de instancia.

La operación DESCONECTA-IM es la desconexión de una instancia molecular de todas las instancias del agregado asociadas a través de un rol específico. Puesto que se

trata de una operación sobre instancias asociadas por un rol, en el esquema correspondiente incluiremos al esquema PRE-Inst-Rol. No será necesario ninguna precondición extra pues no habrá nuevos parámetros.

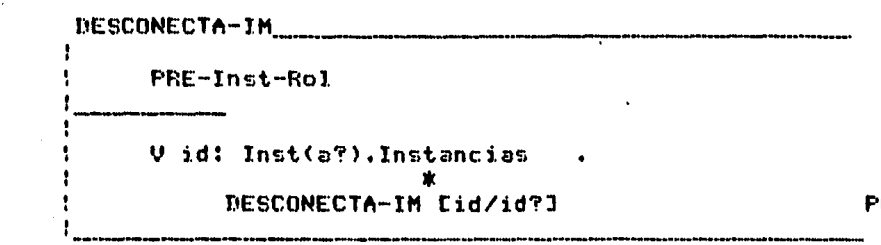
La siguiente gráfica muestra un ejemplo del efecto de la operación.



Para la formalización del efecto de esta operación es conveniente notar que realizar la operación DESCONECTA-IM* es equivalente a efectuar DESCONECTA-IM para todo identificador de instancia (id) que pertenezca al agregado a? y que esté asociado con i?. Este enfoque de la operación nos ayudará a simplificar mucho la especificación formal ya que, al definir a DESCONECTA-IM en función de DESCONECTA-IM*, automáticamente estaremos considerando a DESCONECTA-IM como operación total pues DESCONECTA-IM* lo es. Además, cada aplicación de DESCONECTA-IM* correspondiente a cada id implicará el chequeo de precondiciones y postcondiciones ya definidos en esa operación.

A continuación presentamos el esquema que especifica

formalmente a DESCONECTA-IM.



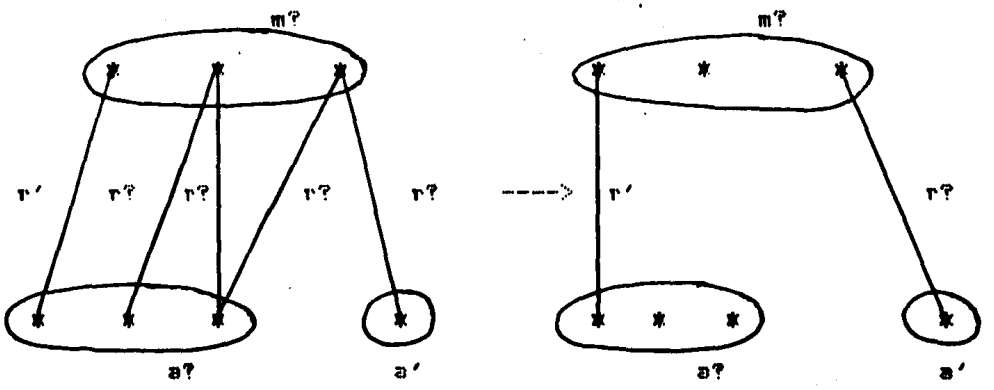
VI.3.3 OTRAS OPERACIONES DE DESCONEXION.

En este apartado especificaremos varias operaciones más de desconexión que serán definidas en función de otras operaciones.

Llamaremos DESCONECTA-MOL a la operación total de desconexión de todas las instancias de una molécula de todas las instancias de un agregado que estén asociadas a través de un rol específico.

En el esquema correspondiente incluiremos como componentes a m?, que es la clase de las instancias moleculares a desconectar; a? , que es la clase del agregado del cual se quiere desconectar, y r? que es el rol a través del cual se desconectará.

Ejemplificaremos el efecto de DESCONECTA-MOL en la siguiente gráfica.



Esta operación es equivalente a efectuar DESCONECTA-IM para toda instancia i de m? por lo que el esquema correspondiente será el siguiente.

DESCONECTA-MOL

δ BDM
I
m?: NOM-CLASE

a?: NOM-CLASE

r?: NOM-ROL

∀ i: rng Inst(m?).IC .

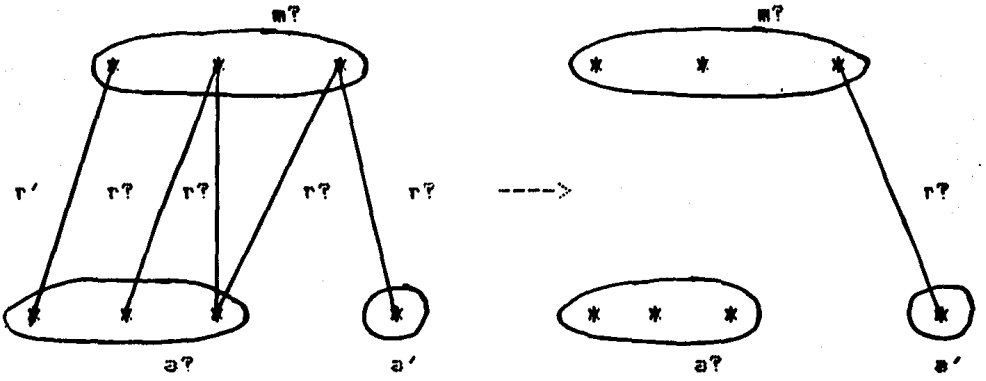
DESCONECTA-IM [i/i?] POS 15

Otra operación total que especificaremos será la desconexión de todas las instancias de una molécula de todas las instancias de un agregado bajo todos los roles, lo que producirá la completa separación de las dos clases involu-

crádas (molécula y agregado) como veremos adelante.

Llamaremos DESCONECTA-AGR a dicha operación. En el esquema que especificará a la operación, incluiremos a los componentes m? y a? . Ya no será necesario indicar a través de qué rol se efectuará la desconexión puesto que se trata de afectar a todos los roles.

La siguiente gráfica ejemplifica la operación. Note que la gráfica inicial es la misma del ejemplo anterior y observe la diferencia en el efecto.



DESCONECTA-AGR es equivalente a efectuar DESCONECTA-MOL para todo rol a través del cual se agregue a?, lo cual queda especificado en la postcondición del siguiente esquema.

DESCONECTA-AGR

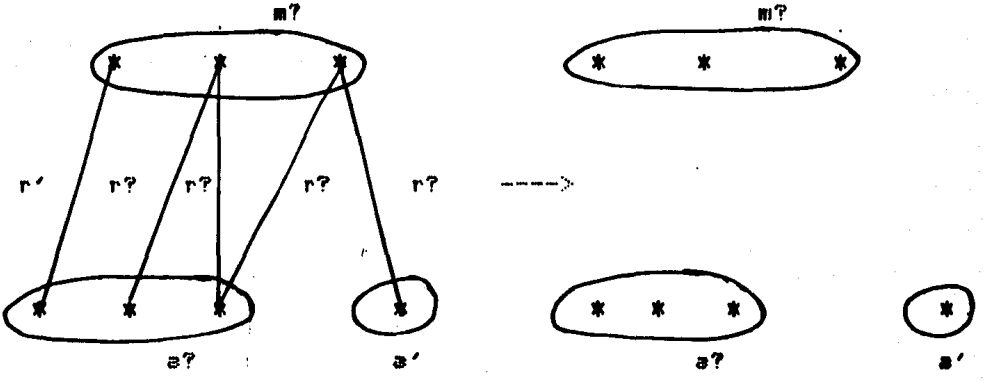
δ BDM
I
m?: NOM-CLASE

a?: NOM-CLASE

∀ r : Esq(m?).Roles .

La siguiente operación total que definiremos será llamada DESCONECTA-AGREGADOS. Esta operación es la desconexión de todas las instancias de una molécula de las instancias de todos sus agregados bajo todos los roles.

Solo habrá un componente en el esquema (además del indicador de posible cambio en IBDM) y será m?, el nombre de la molécula. El efecto de la operación se muestra partiendo de la misma gráfica inicial del ejemplo anterior.



El efectuar DESCONECTA-AGREGADOS es equivalente a efectuar DESCONECTA-AGR para todo agregado de la molécula m?.

El siguiente es el esquema que especifica a DESCONECTA-AGREGADOS.

DESCONECTA-AGREGADOS

δ BDM
I

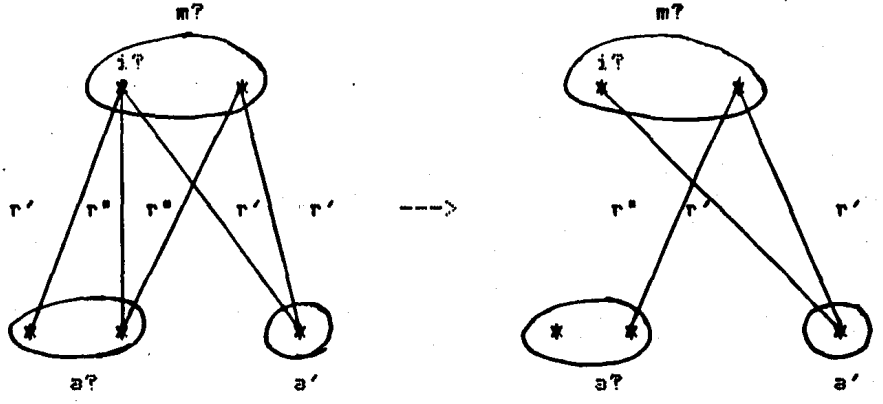
m?: NOM-CLASE

∀ a : Esq(m?).Agregados .

DESCONECTA-AGR [a/a?] POS17

Otra operación a definir será la desconexión de una instancia molecular de todas las instancias de un agregado asociado a ella a través de cualquier rol, le llamaremos DESCONECTA-I-AGR.

Los componentes necesarios en el esquema serán los tres parámetros m?, a?, i? que son de los tipos acostumbrados. El efecto de la operación se ilustra en la gráfica siguiente.



DESCONECTA-I-AGR es equivalente a efectuar DESCONECTA-IM para todo rol a través del cual i? tenga instancias agregadas de a?. La especificación formal se muestra en el esquema.

DESCONECTA-I-AGR

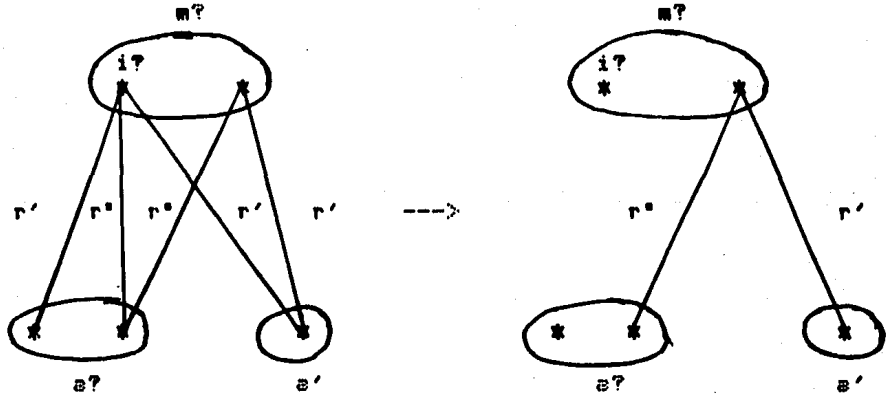
```
δ BDM
  I
m?: NOM-CLASE

a?: NOM-CLASE

i?: NOM-ATR --//--> VALOR
```

∀ r : Esq(m?).Roles .

DESCONECTA-IM [r/r?] POS18

La última operación de desconexión que formalizaremos será la desconexión de una instancia molecular de todas las instancias de todos sus agregados bajo cualquier rol, le llamaremos DESCONECTA-I-AGREGADOS. El esquema tendrá como componentes a m? e i? no será necesario el parámetro de entrada a?.

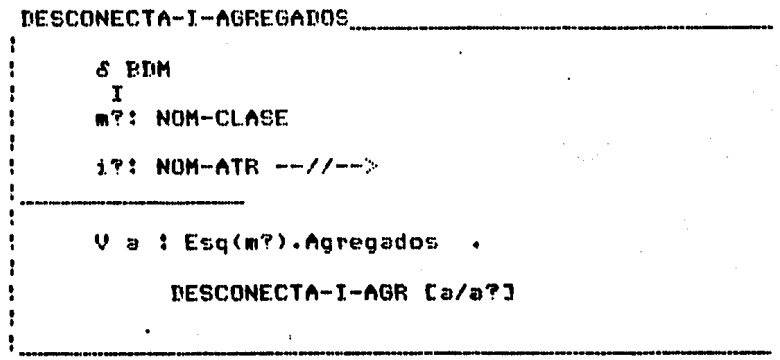
La siguiente gráfica ilustra el efecto de la operación.



DESCONECTA-I-AGREGADOS es equivalente a efectuar DESCONECTA-I-AGR para todo agregado de la molécula m?.

El esquema que especifica formalmente lo anterior es

el siguiente.



VI.4 CONEXION.

En esta sección especificaremos formalmente algunas operaciones de conexión. En general, el efecto de dichas operaciones será la conexión de instancias de una molécula con instancias de agregados de esa molécula.

Recordemos que el esquema PRE-Inst-Rol especifica los parámetros básicos para una operación sobre instancias asociadas por un rol y las precondiciones que dichos parámetros deben satisfacer. Puesto que la conexión operará sobre instancias que serán asociadas a través de un rol, el esquema PRE-Inst-Rol será utilizado para especificar algunas de las operaciones de conexión.

VI.4.1 CONECTA INSTANCIA CON INSTANCIA.

A la operación exitosa de conectar una instancia molecular con una instancia de un agregado a través de un rol le hemos llamado CONECTA-I*. Note que el nombre está marcado con asterisco ya que se trata de una operación auxiliar no accesible al usuario que será usada mas adelante para definir operaciones de conexión a nivel de usuario.

En el esquema de CONECTA-I* incluiremos a PRE-Inst-

Rol, cuyos parámetros de entrada son, como ya hemos visto: la instancia molecular i? que se va a conectar, la clase m? de i?, el rol r? a través del cual se hará la conexión, el agregado a? al cual pertenece la instanica con la que se conectará i?.

Será necesario además, incluir otro componente en CONECTA-I*, id?, el identificador de la instancia de a? que será conectado con i?.

Puesto que todos los componentes mencionados hasta ahora también serán incluidos en los esquemas de operación no exitosa de conexión entre instancias, es conveniente definir un esquema que contenga estos componentes y las precondiciones, para abreviar la especificación, le llamaremos MARCO-Conecta-I*.

MARCO-Conecta-I*

PRE-Inst-Rol

id?: ID-INST

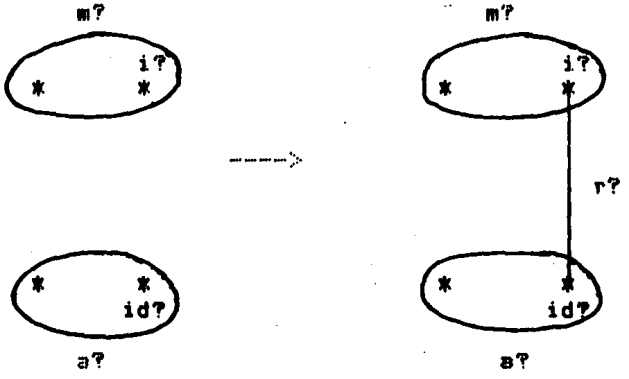
id? ∈ Inst(a?).Instancias PRE13

Con este esquema auxiliar ya definido, bastará incluir MARCO-Conecta-I* en el esquema CONECTA-I* para tener definidos todos los parámetros de entrada y salida necesarios y las precondiciones básicas que éstos deben satisfacer.

En CONECTA-I* incluiremos además una precondición muy importante. Ella garantizará que la operación se e-

fectúe exitosamente solo en el caso que no haya un condicionamiento sobre el rol r? y el agregado a? que lo impida. Es decir, debe cumplirse que la evaluación de la condición (si es que la hay) sea igual a VERDADERO. Esta precondición establecerá que no está definida en Esq(m?) ninguna condición sobre r? para a? o que la evaluación de la condición en Ambiente[id,id?] es igual a VERDADERO, donde id es un identificador de instancia asociado a i?.

La operación CONECTA-I* tiene el efecto que se muestra en el siguiente ejemplo gráfico.



A continuación se presenta el esquema de CONECTA-I* en el que se incluye la precondición que establece que la asociación de instancias está permitida y la postcondición que formaliza el efecto de la operación.

CONECTA-I*

MARCO-Conecta-I*

[(m?,r?,a?) ∈ dom Def-Cond/at]

&

[∀ id: Iden-Asoc(m?,i?) .

Evalúa (Ambiente[id,id?])(Def-Cond/at(m?,r?,a?)) =
expb
VERDADERO

Donde:

Ambiente[id,id?] = Ambiente-Inst[id] U

Ambiente-Inst[id?]

Ambiente-Inst[id] =

{ m?.at !--> val / (m?,id,at) ∈ dom Valor-Inst

& val = Valor-Inst(m?,id,at) }

Ambiente-Inst[id?] =

{ a?.at !--> val / (a?,id?,at) ∈ dom Valor-Inst

& val = Valor-Inst(a?,id?,at) } PRE19

[rep! = "OK"

&

Inst' = Inst @ { m? !--> I-Nueva }] POS20

Donde

I-Nueva = Inst(m?)/R

I-Nueva.R = Inst(m?).R @

{r? !--> (Mapeo-Agreg(m?,r?) @

{a? !--> Mapeo-Nuevo})}

donde

Mapeo-Nuevo = { id !--> id?/

id ∈ Iden-Asoc(m?,i?)} U

Mapeo(m?, r?, a?)

PRE19: Condición sobre atributos para roles válida

para esa asociación de instancias.

POS20: Conexión de la instancia I? con la instancia del agregado con identificador id?.

Hemos especificado ya la operación exitosa de conexión de dos instancias; para especificar la operación total de conexión entre dos instancias, a la que llamaremos CONECTA-IM*, es necesario considerar los casos de operación no exitosa. Ya hemos presentado los esquemas MOLECULA-NoDef, AGREGADO-NoDef, ROL-NoDef, AGR-NoRol, INSTANCIA-NoDef, ATR-NoPrim e ID-NoDef, pero nos falta especificar el caso en el que la asociación de instancias falla, es decir, el caso en el que la evaluación de la condición en el Ambiente para las dos instancias es igual a FALSO, a este esquema le llamaremos COND/AT-Falsa.

COND/AT-Falsa

MARCO-Conecta-I*

[∀ id: Iden-Asoc(m?,i?) .

Evalúa (Ambiente[id,id?])(Def-Cond/at(m?,r?,a?)) =
expb
FALSO

Donde:

Ambiente[id,id?] = Ambiente-Inst[id] ∪
Ambiente-Inst[id?]

Ambiente-Inst[id] =

{ m?.at !--> val / (m?,id,at) ∈ dom Valor-Inst
& val = Valor-Inst(m?,id,at) }

```
        Ambiente-Inst[id?] =
        { a?.at |--> val / (a?,id?,at) ∈ dom Valor-Inst
                        & val = Valor-Inst(a?,id?,at) } ▷ PRE20

    rep!= "Condición sobre atributos falsa"   POS
```

La operación total de conexión de una instancia molecular con una instancia de un agregado a través de un rol queda definida en la forma siguiente.

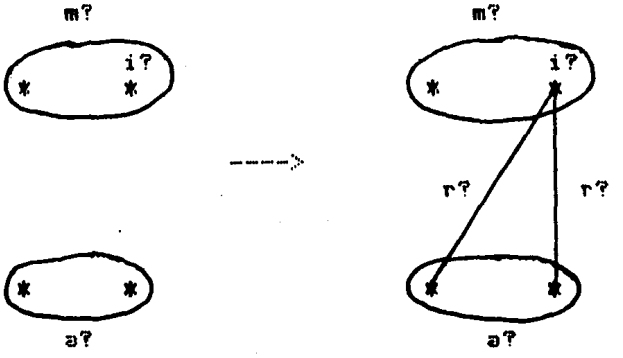
$$\text{CONECTA-IM}^{*} \equiv \text{CONECTA-I}^{*} \vee \text{MOLECULA-NoDef} \vee \text{AGREGADO-NoDef} \vee \text{ROL-NoDef} \vee \text{AGR-NoRol} \vee \text{INSTANCIA-NoDef} \vee \text{ATR-NoPrim} \vee \text{ID-NoDef} \vee \text{COND/AT-Falsa}$$

VI.4.2 OTRAS OPERACIONES DE CONEXION.

En seguida definiremos algunas operaciones totales de conexión que serán especificadas en función de operaciones previamente definidas.

La operación de conexión de una instancia molecular con todas las instancias de un agregado a través de un rol será llamada CONECTA-IM. Como esta operación se efectúa sobre instancias que serán asociadas bajo un rol, incluiremos en el esquema a PRE-Inst-Rol y de esta forma quedarán definidos los parámetros y las precondiciones que deben satisfacer.

El efecto de la operación se ejemplifica en la gráfica siguiente.



En este ejemplo damos por supuesto que con las dos instancias de a? la evaluación de la expresión de condicionamiento de roles sobre atributos (si la hay) es igual a VERDADERO (63).

El efecto de CONECTA-IM es equivalente al efecto que causa efectuar CONECTA-IM* para todo identificador de instancia del agregado a?.

El esquema de CONECTA-IM es el que sigue.

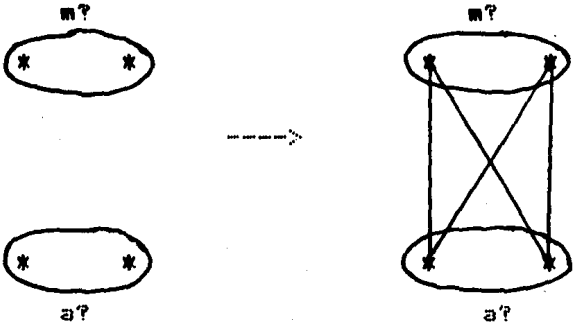
```
CONECTA-IM_____________________________________________
¦
¦     PRE-Inst-Rol
¦_____________________
¦
¦     V id : Inst(a?).Instancias  .
¦                  *
¦          CONECTA-IM [id/id?]                          POS21
¦
¦_____________________________________________________________
```

La siguiente operación que se especificará es la co-

nexión de todas las instancias de una molécula con todas las instancias de un agregado a través de un rol, le llamaremos CONECTA-MOL.

Los parámetros de entrada para esta operación son el nombre de la molécula m?, el agregado a? y el rol r?.

El efectuar CONECTA-MOL produce el efecto que se ejemplifica en la siguiente gráfica, donde todos los roles que aparecen son r?.



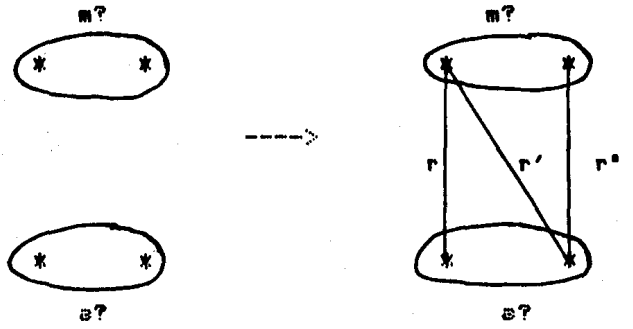
Conecta-MOL es equivalente a efectuar CONECTA-IM para toda instancia de m?. El esquema de la operación es el siguiente.

```
CONECTA-MOL_____________________________________________
|
|     $ BDM
|       I
|     m?: NOM-CLASE
|
|     a?: NOM-CLASE
|
|     r?: NOM-ROL
|_____________________
|
|     V i : rng Inst(m?).IC
|
|          CONECTA-IM [i/i?]                        POS22
|
|_________________________________________________________
```

Otra operación de conexión, a la que llamaremos CONECTA-AGR, es la conexión de todas las instancias de una molécula con todas las instancias de un agregado a través de todos los roles de m?.

Los parámetros de entrada necesarios son m? y a? que son respectivamente la molécula y el agregado.

El efecto de CONECTA-AGR es el que se ejemplifica abajo.



En este ejemplo estamos suponiendo que las otras asociaciones posibles que no se ilustran no satisfacen las condiciones de agregación de r, r' y r°.

La operación CONECTA-AGR es equivalente a la operación CONECTA-MOL efectuada para todo rol r de m?. El esquema de especificación es el siguiente.

CONECTA-AGR

δ BDM
I
m?: NOM-CLASE

a?: NOM-CLASE

```
∀ r : Esq(m?).Roles .
      CONECTA-MOL [r/r?]                                POS23
```

La última operación que definiremos en este apartado es la que hemos llamado CONECTA-AGREGADOS, el efecto de esta operación es el mismo que el de la operación anterior efectuada para todos los agregados de m?, por lo que el único parámetro de entrada necesario es m?.

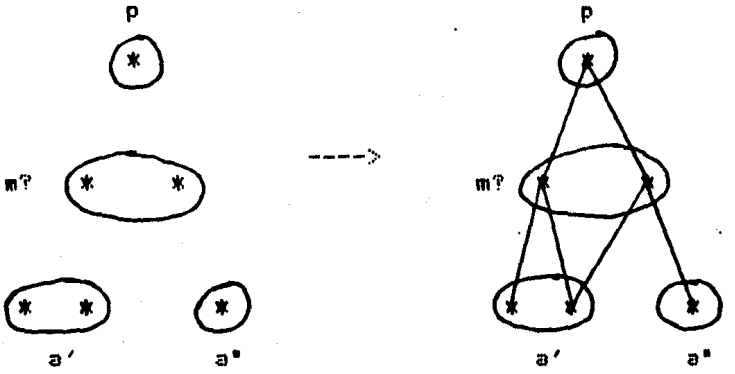
La especificación formal de la operación CONECTA-AGREGADOS se presenta en el siguiente esquema .

```
CONECTA-AGREGADOS
      δ BDM
        I
      m?: NOM-CLASE
      ∀ a : Esq(m?).Agregados .
            CONECTA-AGR [a/a?]                          POS24
```

VI.4.3 AGREGACION.

Llamaremos AGREGACION a la operación de conectar todas las instancias de una molécula m? con todas sus instancias "padre" y con todas sus instancias "hijas" pertenecientes a los agregados de la molécula.

El único parámetro de entrada necesario para la AGREGACION es m?. El efecto de la operación se muestra en el ejemplo gráfico que sigue.



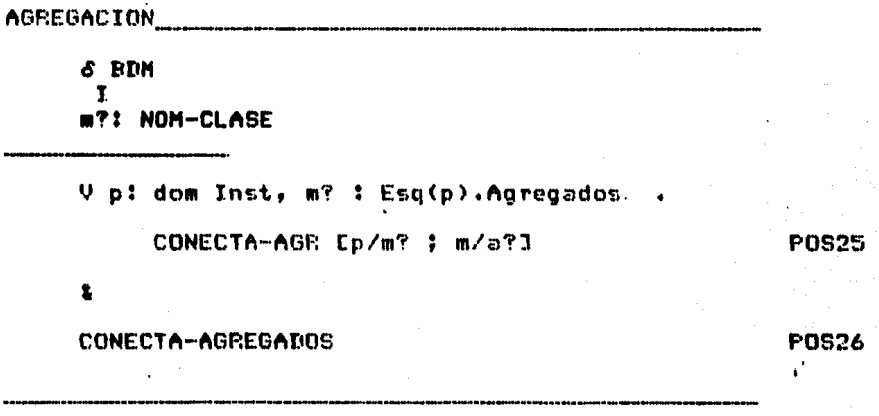
Los roles que aprarecen en la gráfica pueden ser cualesquiera de los definidos para p o m? respectivamente. En el ejemplo, suponemos que los roles mostrados son todos los que satisfacen las condiciones de agregación.

Se puede observar que la AGREGACION aplicada a m? consta de :

- Conexión de todas las instancias moleculares de clases que tengan entre sus agregados a m? (como p en el ejemplo), con todas las instancias de m? a través de todos los roles.
- Y conexión de todas las instancias moleculares de m? con todas las instancias de todos sus agregados (en el ejemplo a' y a") a través de todos los roles.

Es decir que efectuar la operación AGREGACION es equi-

valente a efectuar la operación CONECTA-AGR para toda clase que tenga entre sus agregados a m?, y a efectuar la operación CONECTA-AGREGADOS para m? . Puesto que ya hemos definido las dos operaciones, resulta fácil especificar la AGREGACION según se presenta en el esquema siguiente.



POS25: Conexión de la molécula a todos los "padres" en los cuales dicha molécula aparezca como agregado.

POS26: Conexión de la molécula con todos sus agregados.

VI.5 BORRADO.

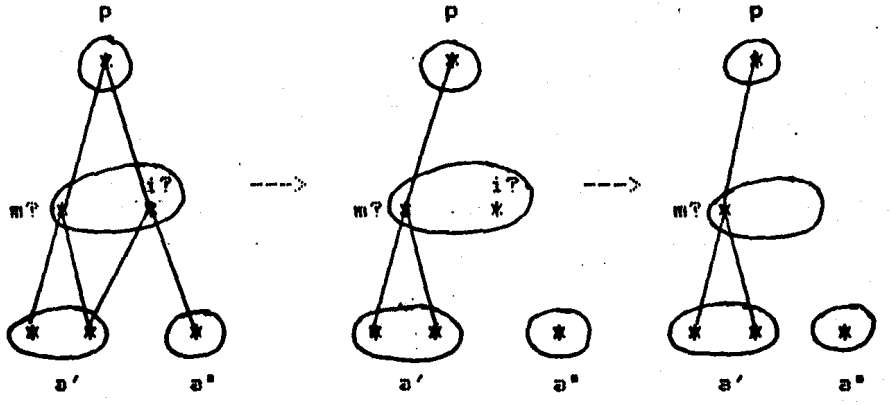
La últimas operaciones que definiremos en el modelo son el borrado o muerte de una instancia y el borrado o muerte de una molécula.

Llamaremos MATA-I a la operación de borrar o dar

muerte a una instancia. Son dos los parámetros necesarios para esta operación: i?, la instancia que se desea borrar y m? la clase de la instancia i?.

Para poder efectuar esa operación es necesario desconectar primero a la instancia i? de todos los roles que lleguen o salgan de ella. Además del efecto anterior de desconexión que causará MATA-I, el otro efecto es la eliminación de la instancia i? de Inst(m?).

La siguiente gráfica muestra un ejemplo del efecto de la operación MATA-I.

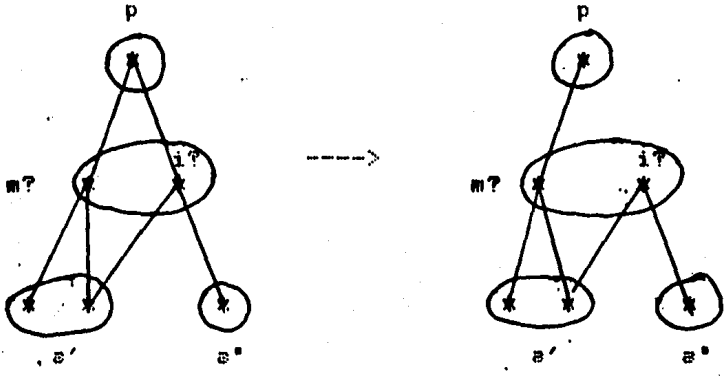


MATA-I es equivalente a la eliminación de la instancia i? previa ejecución de las operaciones

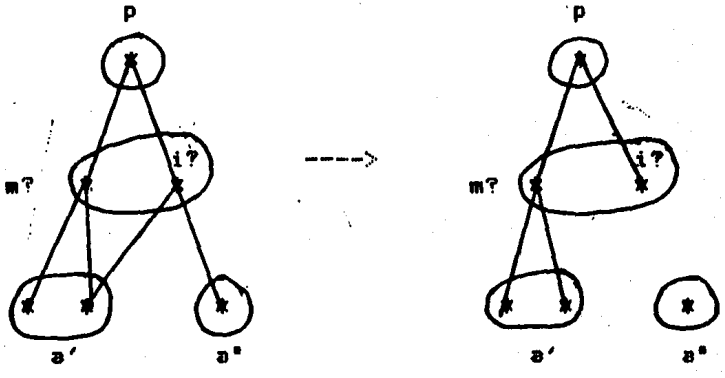
- DESCONECTA-IM* efectuada para toda clase p que contenga entre sus agregados a m?. Para esta operación p será la molécula y m? el agregado. Además DESCONECTA-IM* se efectúa para toda instancia i de p que esté asociada (a través de cualquier rol r) con i?. El parámetro id? para DESCONECTA-IM*

será cualquier identificador asociado a i? (si hay más de un identificador asociado a i?, entonces MATA-I eliminará a todas las instancias correspondientes a dichos identificadores).

En la gráfica inicial del ejemplo anterior, DESCONECTA-IM* actuaría así (suponemos que solo hay una molécula "padre" p).



- DESCONECTA-I-AGREGADOS. Esta operación está definida para los mismos parámetros de MATA-I: m? e i?, la ejecución de esta operación causa la desconexión de todos los roles que salen de i?. En nuestro ejemplo, el efecto es el siguiente.



En el esquema MATA-I siguiente, la última postcondición especifica la eliminación de i? de Inst(m?).

MATA-I

δ BDM
I
m?: NOM-CLASE

i?: NOM-ATR --//--> VALOR

∀ p: dom Inst ; r: Esq(p).Roles; i: rng Inst(p).IC .
(m? : Esq(p).Agregados & id: Iden-Asoc(m?,i?)) ==>
DESCONECTA-I-AGR [p/m?; m?/a?; r/r?; i/i?; id/id?] POS27

DESCONECTA-I-AGREGADOS POS28

Inst' = Inst ⊕ ⟨m? !--> I-Nueva⟩ POS29

Donde

I-Nueva.IC = Iden-Asoc(m?, i?) ⩤ Inst(m?).IC

I-Nueva.R = Inst(m?).R

POS27: Desconecta la instancia de todos sus "padres".

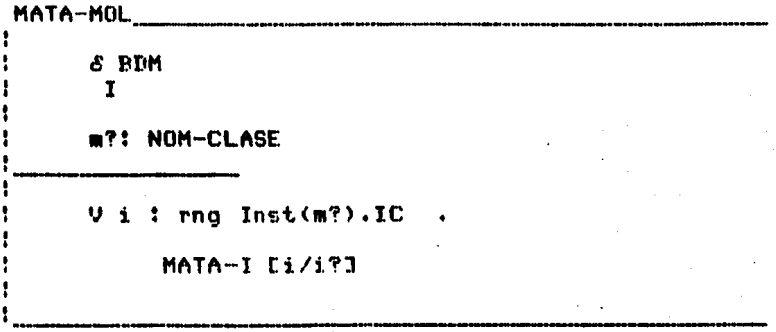
POS28: Desconecta la instancia de todos sus agregados.

POS29: Borra la instancia de la molécula.

Por último, especificaremos la operación que hemos llamado MATA-MOL, la cual borra o da muerte a toda una

molécula.

El único parámetro de entrada necesario es m?. El efecto de MATA-MOL es equivalente al efecto de aplicar MATA-I para toda instancia de m?. El esquema siguiente especifica formalmente a MATA-MOL.



CAPITULO VII.

CONCLUSIONES.

En este capítulo es nuestro interés destacar algunos de los resultados obtenidos a partir del presente trabajo. Puesto que el objetivo del mismo fue la especificación formal de un modelo de datos, resumiremos en primer lugar sus características más importantes. Posteriormente expondremos las ventajas y aportaciones de la especificación formal.

CARACTERISTICAS DEL MODELO.

Recordemos que el desarrollo de nuevos modelos de datos tiene como objetivo responder a las necesidades de los sistemas de información modernos. El modelo orientado a objetos proporciona para esto las siguientes facilidades relacionadas con el manejo de objetos complejos:

- Posibilidad de definición de objetos complejos o moléculas , a través de un mecanismo de agregación molecular que proporciona gran flexibilidad en la estructura de objetos.

- Posibilidad de registro de la semántica que man-

tiene cada agregación en relación con la asociación que se establece entre molécula y agregados.

- Facilidad para el manejo de moléculas disjuntas, no-disjuntas, no- recursivas, y recursivas indistintamente.

Otras características generales son:

- Diferenciación estricta entre instancias y clases de objeto. Esta distinción estricta proporciona al sistema consistencia e integridad sin aumentar su complejidad.

- Diferenciación estricta entre generalización de instancia a clase y generalización de clase a clase.

- Mecanismos de generalización y agregación sumamente flexibles, incluyendo posibilidad del manejo de jerarquías típicas como IS-A y PART-OF entre otras muchas posibilidades.

- Posibilidad de definir reglas de generación, retribución y herencia de atributos en forma precisa, permitiendo su adaptación a múltiples circunstancias.

- Establecimiento de asociaciones a través de roles de agregación que permiten asociaciones de muchos a muchos entre objetos. Un caso particular del

Modelo podría permitir únicamente asociaciones binarias, en esa situación sus roles de agregación serían restringidos a ser binarios.

- Definición intrínseca de restricciones a través del mecanismo de condicionamiento de roles, restringiendo asociaciones a ciertas instancias que satisfagan las condiciones sobre atributos.

- Libertad en la asignación de nombres a atributos y roles. Dado que dichos nombres se encuentran cualificados por su clase, estos pueden repetirse cuando el concepto de asociación o atributo sea el mismo, facilitando y aumentando así el poder expresivo del modelo.

- Control eficiente de instancias de objetos a través de sus identificadores internos únicos generados por el sistema automáticamente, proporcionando mayor integridad.

- Posibilidad de inserción de instancias de objetos siempre que sea necesario, sin necesidad de que dichos objetos tengan valores para un número mínimo de atributos.

- Admite la existencia de instancias que pueden en un momento dado ser iguales, con sus propios identificadores internos y que posteriormente pueden diferir al avanzar un diseño.

Esta revisión breve del modelo orientado a objetos nos muestra que se satisfacen la mayoría de los requisitos que (como vimos en I.3.1) deben satisfacer las bases de datos para poder aplicarse en ambiente CAD y en general, en otras aplicaciones actuales.

APORTACIONES Y RESULTADOS DE LA ESPECIFICACION FORMAL.

La principal aportación de este trabajo es que se ha creado la especificación formal de un modelo para bases de datos orientadas a objetos, la cual proporciona una descripción precisa del modelo, sin ambigüedades. Esto proporciona las siguientes ventajas.

- Se ha construido un modelo que puede funcionar como base para experimentar sobre él, ya sea modificando algunas de sus características, o ya sea aumentando sus capacidades.

- La especificación facilita la tarea de investigación pues se ha podido apreciar cómo el método utilizado permite trabajar aspectos particulares y modificarlos sin que todo el desarrollo tenga que ser modificado.

- La especificación puede ser refinada cada vez más para acercarse a una implementación, sin embargo, aunque el nivel del trabajo realizado es abstracto, es posible observar muchas ventajas que la especi-

ficación proporciona al implementador, como son:

  - La definición de RDM es una guía para la implementación de su lenguaje de definición de datos.
  - La definición de operaciones es una guía para la implementación de un lenguaje de manipulación de datos.
  - Los invariantes describen el estado que el sistema debe mantener y constituyen una guía muy importante para chequeo de restricciones. Puede decirse que los invariantes se convertirán casi automáticamente en reglas de integridad del sistema.

- Se han definido algunos nuevos conceptos en este trabajo y, puesto que se ha realizado la especificación formal de sus característica esenciales, estos podrían ser usados y generalizados en una forma productiva y enriquecedora.

Este trabajo deja varias posibilidades de desarrollo futuro como son las siguientes:

- Posibilidad de definir otras categorías de atributos y otros tipos de condiciones sobre roles.

- Posibilidad de definir otros tipos de restricciones y excepciones , como por ejemplo restricciones sobre los dominios de las variables.

- Definición de otras operaciones como recuperación, actualización, etc. Para este punto se han dado

ya las pautas básicas

- Realización de especificaciones sucesivas cada vez más cercanas a la implementación.

Además de las posibilidades de desarrollo señaladas arriba, creemos que este trabajo puede ser una buena base para una discusión y trabajo sobre modelos del mismo tipo, sobre algunos conceptos expuestos, y, especialmente, sobre la utilidad de aplicar métodos de especificación formal en la definición y estudio de problemas.

Esperamos que el esfuerzo realizado, con sus limitaciones y posibles fallas, motive a los investigadores a trabajar con especificaciones formales.

APENDICE I.

RESUMEN DE LA ESPECIFICACION.

```
[VALOR];

[NOM-DOM];

Val-Dom: NOM-DOM --//--> P VALOR

[NOM-ATR];

DEF-CLASE_____________________________________________
!
!     AP: NOM-ATR --//--> NOM-DOM
!
!     Atr-Prim: F NOM-ATR
!_____________
!
!     rng AP C dom Val-Dom                          I1
!
!     dom AP = Atr-Prim                             I2
!_____________________________________________________
```

I1: Los nombres de dominios deben tener asociados conjuntos de valores.

I2: Definición del conjunto de nombres de atributos primitivos. Todos los atributos primitivos deben tener un dominio asociado.

```
[NOM-ROL];

DEF-CLASE_____________________________________________
!
!     , DEF-CLASE
!
!     Agreg-Rol: NOM-ROL --//--> F NOM-CLASE
!
!     Roles: F NOM-ROL
!
!     Agregados: F NOM-CLASE
```

Roles = dom Agreg-Rol I3

Agregados = U rng Agreg-Rol I4

I3: Definición del conjunto de nombres de roles de la clase. Todos los roles deben tener un conjunto de clases asociadas.

I4: Definición del conjunto de agregados de una clase como la unión distribuida del rango de Agreg-Rol.

EBDM

EBDM

∀ c: dom Esq .

Esq(c).Agregados C dom Esq I5

I5: Todos los agregados deben ser clases definidas.

[ID-INST];

INST-CLASE

IC: ID-INST --//--> (NOM-ATR --//--> VALOR)

Instancias: F ID-INST

dom IC = Instancias I6

I6: Definición del conjunto de identificadores de instancias de una clase.

IBDM

Inst: NOM-CLASE >--//--> INST-CLASE

IBDM

IBDM

$$\forall c_i, c_j : \text{dom Inst} \;.\; (c_i \neq c_j) \Longrightarrow \text{Inst}(c_i).\text{Instancias} \cap \text{Inst}(c_j).\text{Instancias} = \{\} \quad \text{I7}$$

I7: Todos los identificadores de instancias son diferentes.

BDM

EBDM

IBDM

dom Inst $\subseteq$ dom Esq I8

I8: Los nombres de las clases instanciadas son un subconjunto de los nombres de clases definidos.

BDM

Atr-Inst: NOM-CLASE X ID-INST --/--> F NOM-ATR

Valor-Inst: NOM-CLASE X ID-INST X NOM-ATR --/--> VALOR

Dominio: NOM-CLASE X NOM-ATR --/--> NOM-DOM

$$\text{Atr-Inst} = \{ (c,id) \mid\!\!\longrightarrow \text{dom Inst}(c).\text{IC}(id) \;/\; c:\text{dom Inst},\ id:\text{dom Inst}(c).\text{IC} \}$$

$$\text{Valor-Inst} = \{ (c,id,a) \mid\!\!\longrightarrow \text{Inst}(c).\text{IC}(id)(a) \;/\; c:\text{dom Inst},\ id:\text{dom Inst}(c).\text{IC},\ a:\text{dom Inst}(c).\text{IC}(id) \}$$

$$\text{Dominio} = \{ (c,a) \mid\!\!\longrightarrow \text{Esq}(c).\text{AP}(a) \;/$$

c: dom Esq, a: dom Esq(c).AP >

BDM

BDM

∀ (c,id) : dom Atr-Inst .

Atr-Inst(c,id) C Esq(c).Atr-Prim I9

∀ (c,id,a) : Valor-Inst .

Valor-Inst(c,id,a) ∈ Val-Dom (Dominio (c,a)) I10

I9: Consistencia en los nombres de los atributos de las instancias (Los nombres de atributos de una instancia deben ser algunos de los definidos en su clase correspondiente).

I10: Consistencia en los valores asociados a los atributos de las instancias.

INST-CLASE

INST-CLASE

R: NOM-ROL --//--> NOM-CLASE --//-->

(ID-INST <--> ID-INST)

BDM

Mapeo: NOM-CLASE X NOM-ROL X NOM-CLASE -//->

(ID-INST <---> ID-INST)

Mapeo = { (c , r, c) !--> Inst(c).R(r)(c)/
i j i j

c : dom Inst, r: dom Inst(c).R ,

$$c_j : \text{dom rng Inst}(c_i).R(r) \}$$

BDM

BDM

$\forall\, c : \text{dom Inst}$.

$$\text{dom Inst}(c).R \subseteq \text{Esq}(c).\text{Roles} \qquad \text{I11}$$

$\forall\, c : \text{dom Inst}\ ;\ r : \text{dom Inst}(c).R$

$$\text{dom Inst}(c).R(r) \in \text{Esq}(c).\text{Agreg-Rol}(r) \qquad \text{I12}$$

I11: Roles bien definidos.

I12: Las clases asociadas a través de un rol r de la clase c deben estar definidas como agregados de c a través del rol.

BDM

Padres: NOM-CLASE X NOM-ROL X NOM-CLASE -//-> F Id-Inst

Hijos: NOM-CLASE X NOM-ROL X NOM-CLASE -//-> F Id-Inst

$$\text{Padres} = \{ (c_i, r, c_j) \longrightarrow \text{dom Inst}(c_i).R(r)(c_j) = \text{dom Mapeo}(c_i, r, c_j) \;/\; (c_i, r, c_j) : \text{dom Mapeo} \}$$

$$\text{Hijos} = \{ (c_i, r, c_j) \longrightarrow \text{rng Inst}(c_i).R(r).(c_j) = \text{rng Mapeo}(c_i, r, c_j) \;/\; (c_i, r, c_j) : \text{dom Mapeo} \}$$

INST-CLASE

INST-CLASE

$\forall (c_i, r_j, c_j) : \text{dom Padres} \cdot Padres(c_i, r_j, c_j) \subseteq Inst(c_i).Instancias$ I13

I13: Las instancias que son "padre" a través de un rol r_j de la clase c_i deben ser instancias de c_i.

IBDM

IBDM

$\forall (c_i, r_j, c_j) : \text{dom Hijos} \cdot Hijos(c_i, r_j, c_j) \subseteq Inst(c_j).Instancias$ I14

I14: Los "hijos" a traves de un rol r_j de la clase c_i son instancias de clase c_j.

EXPBF[NOM-VAR];

EXP-ATR-GEN ≡ EXPBF[NOM-ATR];

DEF-CLASE

DEF-CLASE

AG: NOM-ATR --//--> NOM-DOM

Def-AG: NOM-ATR --//--> EXP-ATR-GEN

Atr-Gen: F NOM-ATR

Tipo-Atr: NOM-ATR --//--> NOM-DOM

Atributos: F NOM-ATR

∀ exp : rng Def-AG .

Variables(exp) ⊂ Atributos . I15

Atr-Gen = dom AG = dom Def-AG I16

Atr-Gen ∩ Atr-Prim = {} I17

∀ a : Atr-Gen .

AG(a) = Tipo-exp(Def-AG(a)) I18

Tipo-Atr = AP ∪ AG I19

rng Tipo-Atr ⊂ dom Val-Dom I20

Atributos = Atr-Gen ∪ Atr-Prim I21

I15: Las variables de una expresión que define a un atributo generado deben ser atributos de la misma clase.

I16: Definición del conjunto de nombres de atributos generados.

I17: No se permite llamar a los atributos de distintas categorías de la misma forma.

I18: El dominio definido para un atributo generado es igual al dominio del tipo de expresión que lo define.

I19: Definicion del tipo de un atributo (cualquiera).

I20: Todos los dominios correspondientes a los atributos tienen un conjunto de valores asociado.

I21: Definición del conjunto de nombres de atributos (generales) de la clase.

BDM

Dominio: NOM-CLASE X NOM-ATR ----> NOM-DOM

Dominio = { (c, a) |---> Esq(c).Tipo-Atr(a)/

c: dom Esq, a: dom Esq(c).Tipo-Atr }

BDM

depende : NOM-ATR <----> NOM-ATR

a_i depende a_j <===> ($a_i \in$ Atr-Gen) &

$a_j \in$ Variables(Def-AG(a_i))

DEF-CLASE

DEF-CLASE

$\_depende\_^{+}$ $\sqcap$ id NOM-ATR = {} I22

I22: Atributos generados bien definidos.

BDM

BDM

V (c , id, a) : dom Valor-Inst .

[(a : Esq(c).Atr-Gen) ===>

[V a_i : Variables (Esq(c).Def-AG(a)) .

$a_i \in$ dom Ambiente]

&

```
[ Valor-Inst(c, id, a) =
      Evalúa(Ambiente)(Esq(c).Def-AG(a))]  ]   I23
Donde
   Ambiente = Inst(c).IC(id)
```

I23: Los valores de las variables de una expresión que define un atributo generado están definidos en el ambiente y el valor de un atributo generado de una instancia de c está dado por la evaluación de la expresión que lo define en el ambiente.

```
ATRIB
   Cls: NOM-CLASE
   Atr: NOM-ATR

VAR
   Atributo: ATRIB
   Rol: NOM-ROL
```

```
EXP-ATR-HER ≡ EXPBF[VAR]
```

```
DEF-CLASE
   DEF-CLASE
   AH: NOM-ATR --//--> NOM-DOM
   Def-AH: NOM-ATR --//--> EXP-ATR-HER
   Atr-Her: F NOM-ATR
   Atr-Prim  ∩ Atr-Gen ∩ Atr-Her = {}                I17
   Tipo-Atr = AP U AG U AH                           I19
   Atributos = Atr-Prim U Atr-Gen U Atr-Her          I21
   Atr-Her = dom AH = dom Def-AH                     I24
   V a: Atr-Her  .
        AH(a) = Tipo-exp(Def-AH(a))                  I25
```

I24: Definición del conjunto de atributos heredados.

I25: El dominio definido para un atributo heredado es igual al dominio del tipo de expresión que lo define.

```
EBDM
ATRIBUTOS: F ATRIB
-----------------
ATRIBUTOS = { a: ATRIB/ a.Cla ∈ dom Esq  &
                  a.Atr ∈ Esq(a.Cla).Atributos}
```

```
EBDM
Clase : VAR ---> NOM-CLASE
Atrib : VAR ---> NOM-ATR
-----------------
∀ v : VAR  .
     Clase(v) = v.Atributo.Cla
     Atrib(v) = v.Atributo.Atr
```

```
EBDM
EBDM
-----------------
∀ c : dom Esq ;   exp : rng Def_AH;
     v : Variables (exp)  .
        v.Atributo ∈ ATRIBUTOS                    I26
                     &
        v.Rol ∈ Esq(Clase(v)).Roles
```

I26: Toda variable de una expresión que define un atributo heredado se refiere a clases definidas, a atributos de esas clases y a roles de ellas.

BDM

depende-Her : ATRIBUTOS <---> ATRIBUTOS

a_i depende-Her a_j <==>

(a_i.Atr : Esq (a_i.Cla).Atr-Her)

&

(∃ v : Variables (Esq(a_i.Cla).Def-AH(a_i.Atr)) .

a_j.Cla = Clase(v)

&

a_j . Atr = Atrib(v))

BDM

BDM

$$\text{\_depende-Her\_}^{+} \quad \Pi \text{ id ATRIBUTOS} = \{\} \qquad \text{I27}$$

I27: Atributos heredados bien definidos.

EBDM

EBDM

∀ c: dom Esq; exp: rng Esq(c).Def-AH;

v: Variables(exp) .

c ∈ Esq(Clase(v)).Agreg-Rol(v.Rol) I28

I28: Las variables que están en expresiones que definen atributos heredados hacen referencia a clases de objetos "padre" con roles bien definidos para esa clase (herencia a un nivel).

BDM

BDM

∀ c : dom Esq ; exp : rng Esq(c).Def-AH;
v: Variables (exp) .
Mapeo(Clase(v),v.Rol,c) ∈
ID-INST >--//--> ID-INST I29

I29: Ausencia de ambigüedad en instanciación de atributos heredados (Mapeos inyectivos).

BDM

BDM

∀ (c, id, a) : dom Valor-Inst .
(a ∈ Esq(c).Atr-Her) ===>
[∀ v: Variables(exp-AH) ==>
$(Clase(v), id^{-1}(v), Atrib(v)) \in dom\ Valor\text{-}Inst$
&
[Valor-Inst(c,id,a) = Evalúa(Ambiente)(exp-AH) I30
Donde
Ambiente: VAR --//--> VALOR
dom Ambiente = Variables(exp-AH)

```
        ∀ v: dom Ambiente
              Ambiente(v) =
                                        -1
              Valor-Inst(Clase(v), id    (v), Atrib(v)) ]
    Donde
    exp-AH = Esq(c).Def-AH(a)
      -1          -1
    id   (v) = Mapeo   (Clase(v), v.Rol, c)(id),
```

I30: El valor de un atributo heredado de una instancia de c está dado por la evaluación que lo define en el ambiente creado (Evaluación de atributos heredados).

```
EXP-ATR-RET = EXPBF[VAR];
```

DEF-CLASE

```
    DEF-CLASE
    AR: NOM-ATR --//--> NOM-DOM
    Def-AR: NOM-ATR --//--> EXP-ATR-RET
    Atr-Ret: F NOM-ATR
```

```
    Atr-Prim ∩ Atr-Gen ∩ Atr-Her ∩ Atr-Ret = {}      I17
    Tipo-Atr = AP ∪ AG ∪ AH ∪ AR                     I19
    Atributos =
         Atr-Prim ∪ Atr-Gen ∪ Atr-Her ∪ Atr-Ret      I21
    Atr-Ret = dom AR = dom Def-AR                    I31
    ∀ a: Atr-Ret   .
         AR(a) = Tipo-exp (Def-AR(a))                I32
```

I31: Definición del conjunto de atributos retribuidos.

I32: El dominio definido para un atributo retribuido

es igual al dominio del tipo de expresión que lo define.

EBDM

EBDM

∀ c : dom Esq ; exp : rng Def_AR;

v : Variables (exp) .

v.Atributo ∈ ATRIBUTOS I33

&

v.Rol ∈ Esq(c).Roles

I33: Toda variable de una expresión que define un atributo retribuido se refiere a clases definidas, a atributos de esas clases y a roles de la clase en la que se define el atributo.

BDM

depende-Ret : ATRIBUTOS <---> ATRIBUTOS

a depende-Ret a <==>
 i j

(a .Atr : Esq (a .Cla).Atr-Ret)
 i i
 &

(∃ v : Variables (Esq(a .Cla).Def-AR(a .Atr)
 i i
 a .Cla = Clase(v)
 j
 &

 a . Atr = Atrib(v))
 j

BDM

BDM

$_{depende\text{-}Ret}^{+}$ ∩ id.ATRIBUTOS = {} I34

I34: Atributos retribuidos bien definidos.

EBDM

EBDM

∀ c : dom Esq ; exp : rng Esq(c).Def-AR;

v: Variables (exp) .

Clase(v) ∈ Esq(c).Agreg-Rol(v.Rol) I35

I35: Las variables que están en expresiones que definen atributos retribuidos hacen referencia a clases de objetos agregados a través de v.Rol (Retribución a un nivel).

BDM

BDM

∀ c : dom Esq ; exp : rng Esq(c).Def-AR;

v: Variables (exp) .

Mapeo(c, v.Rol, Clase(v)) ∈

ID-INST--//-->ID-INST I36

I36: Ausencia de ambigüedad en instanciación de atributos retribuidos (Mapeo debe ser función).

BDM

BDM

∀ (c, id, a) : dom Valor-Inst .

(a ∈ Esq(c).Atr-Ret) ===>

[∀ v: Variables(exp-AR) ==>

(Clase(v), id* (v), Atrib(v)) ∈ dom Valor-Inst

&

[Valor-Inst(c,id,a) = Evalúa(Ambiente)(exp-AR) I37

Donde

Ambiente: VAR --//--> VALOR

dom Ambiente = Variables(exp-AR)

∀ v: dom Ambiente

Ambiente(v) =

Valor-Inst(Clase(v), id* (v), Atrib(v))]

Donde

exp-AR = Esq(c).Def-AR(a)

id* (v) = Mapeo(c, v.Rol, Clase(v))(id),

I37: El valor de un atributo retribuido de una instancia de c está dado por la evaluación que lo define en el ambiente creado (Evaluación de atributos retribuidos).

EXPB/AT ≡ EXPBOLBF [ATRIB];

DEF-CLASE

DEF-CLASE

Cond/at: NOM-ROL -//-> (NOM-CLASE -//-> EXPB/AT)

---

dom Cond/at C Roles I38

∀ r: dom Cond/at .

dom Cond/at(r) ⊂ Agreg-Rol(r) I39

I38: Condición establecida sobre roles definidos.

I39: Condición establecida sobre agregado adecuado.

$c.a = \mu \{ v{:}ATRIB / v.Cla = c \ \& \ v.Atr = a \}$

BDM

Def-Cond/at:

NOM-CLASE X NOM-ROL X NOM-CLASE ---> EXPB/AT

$$Def\text{-}Cond/at = \{ (c_i, r, c_j) \mapsto Esq(c_i).Cond/at(r)(c_j) / c_i \in dom\ Esq,\ r: dom\ Esq(c_i).Cond/at) \ \& \ (c_j \in dom\ rng\ Esq(c_i).Cond/at(r)) \}$$

EBDM

EBDM

$$\forall (c_i, r, c_j) : dom\ Def\text{-}Cond/at \ ; \ v \in Variables_{expb}(Def\text{-}Cond/at(c_i, r, c_j)) .$$

$$(Clase(v) \ \& \ Atrib(v) \in Esq(c_i).Atributos_v)$$

$$(Clase(v) \ \& \ Atrib(v) \in Esq(c_j).Atributos) \quad I40$$

I40: Las variables de una expresión que define una

condición sobre atributos sólo pueden referirse a atributos de la clase definida o de su agregado.

BDM

BDM

$\forall (c_i, r_j, c_j) : dom\ Mapeo\ .$
$(c_i, r_j, c_j) \in dom\ Def\text{-}Cond/at \Rightarrow$

$[\ \forall v{:}Variables_{expb}\ (Def\text{-}Cond/at(c_i, r_j, c_j)\ .$
$v \in dom\ Ambiente$

&

$[\ \forall (i_1, i_2) \in Mapeo(c_i, r_j, c_j)\ .$

Evalúa $(Ambiente[i_1, i_2])(Def\text{-}Cond/at(c_i, r_j, c_j))_{expb}$

$= VERDADERO$ I41

Donde

$Ambiente[i_1, i_2] \equiv Ambiente\text{-}Inst[i_1] \cup Ambiente\text{-}Inst[i_2]$

$Ambiente\text{-}Inst[i] \equiv \{c.a \dashrightarrow val / (c,i,a) \in Valor\text{-}Inst \ \&$
$val = Valor\text{-}Inst(c,i,a)\}$

I41: Restricción del Mapeo a las instancias que satisfagan la Cond/at.

OPERACIONES.

I-BDM

BDM

```
    ∀ c : dom Inst  .
          Inst(c).IC = {}
          Inst(c).Instancias = {}
          dom Esq(c).Roles = dom.Inst(c).R
          rng Inst(c).R = { {} }
```

δ BDM
I

```
    bdm, bdm' : BDM
    ---------------
    Esq' = Esq
```

δ BDM
E

```
    bdm, bdm' : BDM
    ---------------
    Inst' = Inst
```

≡BDM

```
    bdm, bdm' : BDM
    ---------------
    Esq' = Esq
    Inst' = Inst
```

BDM

```
    BDM
    Nuevos-Id : F ID-INST
    ---------------
    Nuevos-Id = { id: ID-INST/ ∀ c:dom Inst,
                      id ∉ Inst(c).Instancias }
```

REPORTE ≡ Seq CARACTER

MARCO-Inserta

c?: NOM-CLASE

i?: NOM-ATR --//--> VALOR

rep!: REPORTE

INSERTA-Nuevo

δ BDM
I

MARCO-Inserta

c?: dom Esq PRE1

dom i? C Esq(c?).Atr-Prim PRE2

∀ a : dom i? .
i?(a) ∈ Val-Dom(Dominio(c?,a) PRE3

rep! = "OK" POS1

Inst'=Inst ⊕ {c? !--> I-Nueva} POS2

Donde

I-Nueva = Inst(c?)/IC

I-Nueva.IC = Inst(c?).IC U {idN !-->i?}

donde

idN : Nuevos-Id

PRE1: Clase definida.

PRE2: Los nombres de atributos de la instancia corresponden a nombres de atributos primitivos

definidos.

PRE3: Los valores de la instancia pertenecen a los dominios correctos.

POS1: Reporte de inserción exitosa.

POS2: Se incorpora una nueva instancia a la clase.

CLASE-NoDef

≡BDM

MARCO-Inserta

c? ∉ dom Esq — PRE4

rep!= "Clase no definida" — POS3

PRE4: Clase no definida.

POS3: Reporte de inserción no exitosa.

ATR-NoDef

≡BDM

MARCO-Inserta

∃ a : dom i? .

a ∉ Esq(c?).Atr-Prim — PRE5

rep!= "Atributo no definido" — POS4

PRE5: Atributo no definido.

POS4: Reporte de inserción no exitosa.

VALOR-Incon

≡BDM

c?: NOM-CLASE

i?: NOM-ATR --//--> VALOR

∃ a : dom i? .

i?(a) ∉ Val-Dom(Dominio(c?,a)) PRE6

rep!= 'Valor inconsistente' . POS5

PRE6: Valor inconsistente.

POS5: Reporte de inserción no exitosa.

INSERTA ≡ INSERTA-Nuevo ∨ CLASE-NoDef ∨
ATR-NoDef ∨ VALOR-Incon

BDM

Instancia: NOM-CLASE X ID-INST -//-->
(NOM-ATR -//--> VALOR)

Mapeo-Agreg: NOM-CLASE X NOM-ROL -//-->
(NOM-CLASE -//-> (ID-INST <-->ID-INST))

Iden-Asoc:
NOM-CLASE X (NOM-ATR -//-> VALOR) -//->
F ID-INST

Instancia = { (c, id) !--> Inst(c).IC(id) /
c: dom Inst, id: Inst(c).Instancias}

Mapeo-Agreg = { (c, r) !--> Inst(c).R(r) /
c:dom Inst, r: dom Inst(c).R}

∀ i: (NOM-ATR -//--> VALOR) ;
c: dom Inst .
Iden-Asoc(c,i) = {id: ID-INST/
i C Inst(c).IC(id)}

MARCO-Inst-Rol

δ BDM
I

m?: NOM-CLASE

a?: NOM-CLASE

r?: NOM-ROL

i?: NOM-ATR --//--> VALOR

rep!: REPORTE

PRE-Inst-Rol

MARCO-Inst-Rol

id?: ID-INST

---

m? ∈ dom Inst	PRE7
a? ∈ dom Inst	PRE8
r? ∈ Esq(m?).Roles	PRE9
a? ∈ Esq(m?).Agreg-Rol(r?)	PRE10
] id: Inst(m?).Instancias .	
id ∈ Iden-Asoc(m?,i?)	PRE11
dom i? C Esq(m?).Atr-Prim	PRE12

PRE7: Molécula definida.

PRE8: Agregado definido.

PRE9: Rol definido.

PRE10: Agregado correspondiente a rol definido.

PRE11: Instancia definida.

PRE12: Atributos de la instancia primitivos.

DESCONECTA-I*

PRE-Inst-Rol

id?: ID-INST

id? ∈ Inst(a?).Instancias PRE13

Inst' = Inst ⊕ {m? !--> I-Nueva} POS6

Donde

I-Nueva = Inst(m?)/R

I-Nueva.R = Inst(m?).R ⊕

{r !--> (Mapeo-Agreg(m?,r?) ⊕

{a? !--> Mapeo-Nuevo})}

donde

Mapeo-Nuevo = Mapeo(m?, r?, a?) -

{id !--> id? / id ∈ Iden-Asoc(m?,i?)}

PRE13: Identificador perteneciente al agregado.

POS6: Desconexión de la instancia de instancia.

MOLECULA-NoDef

MARCO-Inst-Rol

id?: ID-INST

m? ∉ dom Inst PRE14

rep!: "Molécula no definida" POS 7

AGREGEGADO-NoDef

MARCO-Inst-Rol

id?: ID-INST

a? ∉ dom Inst PRE15

rep!: "Agregado no definido" POS8

ROL-NoDef

MARCO-Inst-Rol

id?: ID-INST

r? ∉ Esq(m?).Roles — PRE16

rep!: 'Rol no definido' — POS9

AGR-NoRol

MARCO-Inst-Rol

id?: ID-INST

a? ∈ Esq(m?).Agreg-Rol(r?) — PRE17

rep!: 'El agregado no corresponde al rol' — POS10

INSTANCIA-NoDef

MARCO-Inst-Rol

id?: ID-INST

∀ id ∈ Inst(m?).Instancias .

i? ∉ Instancia(m?,id) — PRE18

rep!: 'Instancia no definida' — POS11

ATR-NoPrim

MARCO-Inst-Rol

id?: ID-INST

dom i? ∉ Esq(m?).Atr-Prim — PRE19

rep!: 'Atributos de instancia no primitivos' — POS12

ID-NoDef

MARCO-Inst-Rol

```
    id?: ID-INST
    ------------
    id? ∉ Inst(a?).Instancias                                  PRE20
    rep!: 'Identificador no perteneciente al agregado'POS13
```

$$\text{DESCONECTA-IM} \equiv \text{DESCONECTA-I}^{*} \vee \text{MOLECULA-NoDef} \vee \text{AGREGADO-NoDef} \vee \text{ROL-NoDef} \vee \text{AGR-NoRol} \vee \text{INSTANCIA-NoDef} \vee \text{ID-NoDef} \vee \text{ATR-NoPrim}$$

```
DESCONECTA-IM
    PRE-Inst-Rol
    ------------
    ∀ id: Inst(a?).Instancias  .
                    *
          DESCONECTA-IM [id/id?]                               POS14
```

```
DESCONECTA-MOL
    δ BDM
     I
    m?: NOM-CLASE

    a?: NOM-CLASE

    r?: NOM-ROL
    ------------
    ∀ i: rng Inst(m?).IC  .

          DESCONECTA-IM [i/i?]                                 POS 15
```

```
DESCONECTA-AGR
    δ BDM
     I
    m?: NOM-CLASE

    a?: NOM-CLASE
```

```
¦_____________
¦
¦     V r : Esq(m?).Roles  .
¦
¦          DESCONECTA-MOL [r/r?]                              POS16
¦
¦_____________________________________________________________

DESCONECTA-AGREGADOS_____________________________________________
¦
¦     δ BDM
¦      I
¦     m?: NOM-CLASE
¦_____________
¦
¦     V a : Esq(m?).Agregados  .
¦
¦          DESCONECTA-AGR [a/a?]                              POS17
¦
¦_____________________________________________________________

DESCONECTA-I-AGR:_________________________________________________
¦
¦     δ BDM
¦      I
¦     m?: NOM-CLASE
¦
¦     a?: NOM-CLASE
¦
¦     i?: NOM-ATR --//--> VALOR
¦_____________
¦
¦     V r : Esq(m?).Roles  .
¦
¦          DESCONECTA-IM [r/r?]                               POS18
¦
¦_____________________________________________________________

DESCONECTA-I-AGREGADOS____________________________________________
¦
¦     δ BDM
¦      I
¦     m?: NOM-CLASE
¦
¦     i?: NOM-ATR --//-->
¦_____________
¦
¦     V a : Esq(m?).Agregados  .
¦
¦          DESCONECTA-I-AGR [a/a?]                            POS19
¦
¦_____________________________________________________________

                *
MARCO-Conecta-I ___________________________________________________
```

PRE-Inst-Rol

id?: ID-INST

---

id? ∈ Inst(a?).Instancias PRE13

CONECTA-I *

MARCO-Conecta-I *

[(m?,r?,a?) ∈ dom Def-Cond/at]

&

[∀ id: Iden-Asoc(m?,i?) .

Evalúa (Ambiente[id,id?])(Def-Cond/at(m?,r?,a?)) =
exp b
VERDADERO

Donde:

Ambiente[id,id?] = Ambiente-Inst[id] ∪
Ambiente-Inst[id?]

Ambiente-Inst[id] =

{ m?.at |--> val / (m?,id,at) ∈ dom Valor-Inst
& val = Valor-Inst(m?,id,at) }

Ambiente-Inst[id?] =

{ a?.at |--> val / (a?,id?,at) ∈ dom Valor-Inst
& val = Valor-Inst(a?,id?,at) } PRE19

[rep! = "OK"

&

Inst' = Inst ⊕ { m? |--> I-Nueva }] POS20

Donde

I-Nueva = Inst(m?)/R

I-Nueva.R = Inst(m?).R ⊕

{r? |--> (Mapeo-Agreg(m?,r?) ⊕

```
                        {a? !--> Mapeo-Nuevo})}
        donde
             Mapeo-Nuevo = { id !--> id?/
                             id ∈ Iden-Asoc(m?,i?)} U
                               Mapeo(m?, r?, a?)
```

PRE19: Condición sobre atributos para roles válida para esa asociación de instancias.

POS20: Conexión de la instancia I? con la instancia del agregado con identificador id?.

COND/AT-Falsa

```
     MARCO-Conecta-I*

     [ V id: Iden-Asoc(m?,i?)  .
     Evalúa  (Ambiente[id,id?])(Def-Cond/at(m?,r?,a?)) =
          expb
                                                    FALSO
     Donde:
          Ambiente[id,id?] = Ambiente-Inst[id] U
                             Ambiente-Inst[id?]
          Ambiente-Inst[id] =
          { m?.at !--> val / (m?,id,at) ∈ dom Valor-Inst
                       & val = Valor-Inst(m?,id,at) }
          Ambiente-Inst[id?] =
          { a?.at !--> val / (a?,id?,at) ∈ dom Valor-Inst
                       & val = Valor-Inst(a?,id?,at) } PRE20

     rep!= 'Condición sobre atributos falsa'   POS15
```

```
CONECTA-IM* ≡ CONECTA-I*  v  MOLECULA-NoDef  v
```

AGREGADO-NoDef v ROL-NoDef v

AGR-NoRol v INSTANCIA-NoDef v

ATR-NoPrim v ID-NoDef v

COND/AT-False

CONECTA-IM

PRE-Inst-Rol

∀ id : Inst(a?).Instancias .
*
CONECTA-IM [id/id?] POS21

CONECTA-MOL

δ BDM
I
m?: NOM-CLASE

a?: NOM-CLASE

r?: NOM-ROL

∀ i : rng Inst(m?).IC

CONECTA-IM [i/i?] POS22

CONECTA-AGR

δ BDM
I
m?: NOM-CLASE

a?: NOM-CLASE

∀ r : Esq(m?).Roles .

CONECTA-MOL [r/r?] POS23

CONECTA-AGREGADOS

δ BDM
I

```
    m?: NOM-CLASE
_____________________

    V a : Esq(m?).Agregados  .

         CONECTA-AGR [a/a?]                                  POS24

_____________________________________________________________

AGREGACION___________________________________________________

    δ BDM
     I
    m?: NOM-CLASE
_____________________

    V p: dom Inst, m? : Esq(p).Agregados  .

         CONECTA-AGR [p/m? ; m/a?]                           POS25

    &

    CONECTA-AGREGADOS                                        POS26

_____________________________________________________________
```

POS25: Conexión de la molécula a todos los 'padres' en los cuales dicha molécula aparezca como agregado.

POS26: Conexión de la molécula con todos sus agregados.

```
MATA-I_______________________________________________________

    δ BDM
     I
    m?: NOM-CLASE

    i?: NOM-ATR --//--> VALOR
_____________________

    V p: dom Inst ; r: Esq(p).Roles; i: rng Inst(p).IC  .

     ( m? : Esq(p).Agregados  &  id: Iden-Asoc(m?,i?) ) ==>

         DESCONECTA-I-AGR [p/m?; m?/a?; r/r?; i/i?; id/id?]

                                                             POS27

    DESCONECTA-I-AGREGADOS                                   POS28
```

```
Inst' = Inst @ <m? |--> I-Nueva>                                POS29

Donde

     I-Nueva.IC = Iden-Asoc(m?, i?) ◁
                  Inst(m?).IC

     I-Nueva.R = Inst(m?).R
```

POS27: Desconecta la instancia de todos sus 'padres'.

POS28: Desconecta la instancia de todos sus agrega-
dos.

POS29: Borra la instancia de la molécula.

```
MATA-MOL

     δ BDM
      I

     m?: NOM-CLASE

     ∀ i : rng Inst(m?).IC  .

          MATA-I [i/i?]                                         POS30
```

APENDICE II.

ESPECIFICACION DEL LENGUAJE DE EXPRESIONES.

En este apéndice formalizamos el lenguaje de expresiones bien formadas que se utiliza para la definición de atributos en el modelo orientado a objetos.

Se presenta la sintáxis abstracta del lenguaje, es decir, la estructura de las expresiones del lenguaje, y la especificación denotativa de su semántica, es decir, se definen algunas funciones que describen el significado de las expresiones del lenguaje (64).

En base al lenguaje de expresiones bien formadas se define un lenguaje de expresiones boleanas que sirve para establecer condicionamientos sobre roles. Para extender el lenguaje de expresiones bien formadas, primero se incluye la definición de predicados para finalemente definir el lenguaje de expresiones boleanas.

LENGUAJE DE EXPRESIONES BIEN FORMADAS.

El lenguaje que se describe a continuación está parametrizado por el conjunto NOM-VAR (nombres de variables), que se introduce en seguida.

[NOM-VAR];

También haremos referencia a dos conjuntos ya introducidos en el modelo: VALOR y NOM-DOM que son respectivamente el conjunto de valores y el conjunto de posibles nombres de dominios.

[VALOR];

[NOM-DOM];

En el siguiente esquema se especifican las operaciones básicas del lenguaje. Recuerde que la definición de VALOR queda abierta en el modelo, dependiendo de la forma en que se vayan a almacenar los datos. Sin embargo, para poder definir las operaciones del lenguaje supondremos que los valores son secuencias de caracteres.

VALOR ≡ Seq CARACTER

OPERACIONES

Head: VALOR ---> VALOR
Last: VALOR ---> VALOR
Concat: Seq VALOR ---> VALOR
Suma: VALOR X VALOR ---> VALOR
Resta: VALOR X VALOR ---> VALOR
Mult: VALOR X VALOR ---> VALOR

Nota Técnica: Se muestran solo algunas operaciones a manera de ejemplo,pero podrían definirse otras.

La definición sintáctica del conjunto de operadores OP es la siguiente.

OP : : = Head / Concat / Suma / Resta / Mult

Definiremos ahora una función, que llamaremos Aridad, que será auxiliar para definir más adelante lo que es una expresión bien formada. Esta función indicará para cuantos parámetros está bien definida cada operación.

Aridad: OP ----> N

Aridad = { Head !--> 1, Last !--> 1, Concat !--> 0,
Suma !--> 2, Resta !--> 2, Mult !--> 2 }

En general, puede darse el caso de operaciones con aridad variable, cuando esto suceda diremos que la aridad de esa operación es 0 (la concatenación tiene definida una aridad variable).

La siguiente función define cual será el significado de una operación aplicada sobre una secuencia de valores, será llamada S_op.

```
S_op: OP ----> Seq VALOR --/--> VALOR

V v1, v2: VALOR  .
  ( v1, v2 ∈ Val-Dom(CUERDA)  .
     S_op Head [v1] = head v1
     S_op Last [v1] = last v1
     S_op Concar [v1,v2] = v1 ^ v2 )
  &
  ( v1, v2 ∈ Val-Dom(ENTERO)  .
    S_op Suma [v1,v2] = v1 + v2
    S_op Resta [v1,v2] = v1 - v2
     S_op Mult [v1,v2] = v1 * v2  )
```

Nota Técnica: Suponemos que CUERDA y ENTERO son nombres de dominios definidos. Recuerdar que la signatura de la función que nos da los valores que pertencen a algún dominio es: Val-Dom: NOM-DOM ---> P VALOR.

Definiremos ahora el tipo llamado EXP, que servirá como base para definir en el siguiente esquema a las expresiones bien formadas. Utilizaremos la siguiente notación.

A : : = B << C >>

Esto significa que se define una función B tal que B: C ----> A .

EXP : : = id-var << NOM-VAR >> / val << VALOR >> /
aplica << OP X Seq EXP >> /

aplica-dist << OP X Seq EXP >>

Además, será necesario tener una función que determine cual es el tipo de la expresión. En seguida definimos una función para cada uno de los casos posibles de EXP.

```
tipo-id-var: NOM-VAR ----> NOM-DOM
tipo-val: VALOR ----> NOM-DOM
tipo-op: OP ----> NOM-DOM
tipo-exp: EXP ----> NOM-DOM
```

```
∀ var: VAR  .
     tipo-exp(id-var(var)) = tipo-id-var(var)
∀ v: VALOR  .
     tipo-exp(val(v)) = tipo-val(v)
∀ op: OP; s: Seq EXP  .
     tipo-exp(aplica(op,s)) = tipo-op(op)
     tipo-exp(aplica-dist(op,s)) = tipo-op(op)
```

El siguiente esquema especifica el conjunto de expresiones bién formadas, al que llamaremos EXPBF.

```
EXPBF:  P EXP
```

```
∀ var: NOM-VAR  .
     id-var(var) ∈ EXPBF
```

```
V v: VALOR    .
      val(v) ∈ EXPBF
V op: OP ; s: Seq EXP   .
      aplica(op,s) ∈ EXPBF   <===>
      ( V e   ∈ rng s     .
            i
            tipo-exp(e ) = tipo-op(op)    &
                       i
            e    ∈  EXPBF )
             i
      &
      ( # s = Aridad(op) )
V op: OP ; s: Seq EXP
      aplica-dist(op,s) ∈ EXPBF   <===>
      ( V e   ∈ rng s     .
            i
            tipo-exp(e ) = tipo-op(op)    &
                       i
            e    ∈  EXPBF )
             i
      &
      ( Aridad(op) = 0 )
```

Para poder evaluar una expresión es necesario definir una función que al aplicarle una expresión nos de el conjunto de nombres de variables de la expresión .

```
Variables: EXPBF ----> F NOM-VAR
V var: VAR   .
      Variables(id-var(var)) = {var}
V v: VALOR     .
```

```
        Variables(val(v)) = {}

    ∀ op: OP; s: Seq EXP    .

        Variables(aplica(op,s)) = ∪ {Variables(e ) /
                                      i
                                  e  ∈ rng s }
                                   i

                          =Variables(aplica-dist(op,s)
```

También es necesario definir la función Ambiente (función que asigna a cada nombre de variable un valor y definir una relación que indica cuando el Ambiente contiene en su dominio a todas las varibles de la expresión, ya que solo en ese caso dicha expresión podrá ser evaluada.

Ambiente ≡ NOM-VAR ----> VALOR

```
    _Contiene_ : Ambiente <----> EXPBF

    ∀ a: Ambiente ; e: EXPBF    .

        a Contiene e     <====>

            Variables(e) ⊂ dom a
```

La evaluación de una expresión bien formada queda establecida en el siguiente esquema con la especificación de la función Evalúa.

```
Evalúa: Ambiente -----> EXPBF --/--> VALOR
```

```
V var: VAR  .
     Evalúa (a) (id-var(var)) = a (var)
V v: VALOR    .
     Evalúa (a) (val(v)) = v
V op: OP; s: Seq EXP   .
     Evalúa (a) (aplica(op,s)) =
                          S_op (op) (s ; Evalúa(a) )
     <===>  V e : rng s     .  a Contiene e
               i                          i
V op: OP; s: Seq EXP   .
     Evalúa (a) (aplica-dist(op,s)) =
                          S_op (op) (s ; Evalúa(a) )
     <===>  V e : rng s     .  a Contiene e
               i                          i
```

EXTENSION DEL LENGUAJE.

Ahora extenderemos el lenguaje de expresiones bien formadas incluyendo el conjunto de predicados. Como los predicados son funciones boleanas, es necesario introducir el conjunto BOLEANO.

BOLEANO :: = VERDADERO / FALSO

PREDICADOS

```
Igual: VALOR X VALOR ----> BOLEANO
```

```
Menor: VALOR X VALOR ----> BOLEANO
Mayor: VALOR X VALOR ----> BOLEANO
Diferente: VALOR X VALOR ----> BOLEANO
```

Llamaremos PRED al conjunto de símbolos de predicados.

PRED :: = Igual / Menor / Mayor / Diferente

Vamos a extender el concepto de aridad para que indique para cuantos parámetros está bien definido un predicado.

```
Aridad: OP U PRED ----> N
V p: PRED .
    Aridad (p) = 2
```

La función S_pred que se define en seguida establece el significado de cada uno de los predicados.

```
S_pred: PRED ----> Seq VALOR --/--> BOLEANO
V v1, v2: VALOR .
```

```
S_pred Igual [v1,v2] = VERDADERO si v1=v2
                     = FALSO  si v1<>v2
S_pred Menor [v1,v2] = VERDADERO si v1<v2
                     = FALSO  si v1≥v2
S_pred Mayor [v1,v2] = VERDADERO si v1>v2
                     = FALSO si v1≤v2
S_pred Diferente [v1,v2] = VERDADERO si v1<>v2
                     = FALSO si v1=v2
```

Habiendo incluido ya el concepto de predicado, podemos ahora introducir las operaciones lógicas que nos permitirán convinar predicados y construir el conjunto de expresiones boleanas bien formadas, al cual llamaremos EXPBOLBF.

OPERACIONES LOGICAS

```
And: BOLEANO X BOLEANO ----> BOLEANO
Or: BOLEANO X BOLEANO ----> BOLEANO
Imp: BOLEANO X BOLEANO ----> BOLEANO
Not: BOLEANO ----> BOLEANO
```

La definición sintáctica del conjunto de operadores lógicos es la siguiente.

```
OPlog :: =  And / Or / Imp / Not
```

Una vez mas extenderemos el concepto de aridad para establecer el número de parámetros que debe tener una operación lógica.

```
Aridad: OP U PRED U OPlog ----> N
-------

V op: OPlog    .
      op <> Not ==>
           Aridad (op) = 2
      op = Not  ==>
           Aridad (op) = 1
```

El significado de una operación lógica está dado por la función S_oplog que se define en seguida.

```
S_oplog: OPlog ----> Seq BOLEANO --/--> BOLEANO
--------

V b1, b2: BOLEANO  .
      S_oplog And [b1,b2] =  b1 & b2
      S_oplog Or [b1,b2] = b1 v b2
      S_oplog Imp [b1,b2] =  b1 ==> b2
      S_pred Not [b1] = ~b1
```

Definiremos ahora qué son las expresiones boleanas,

llamadas EXPBOL, para después definir en función de ellas a las expresiones booleanas bien formadas.

```
EXPBOL : : =  valBOL << BOLEANO >> /
              pred << PRED X Seq EXPBF >> /
              aplicaBOL << OPlog X Seq EXPBOL >>
```

Si se deseara definir operadores lógicos con aridad variable se tendría que haber incluido, como en el caso de la definición sintáctica de EXP, a una función aplica-distBOL .

Ahora, ya podemos presentar la especificación de EXPBOLBF.

```
EXPBOLBF:  F EXPBOL

V b: BOLEANO   .
     val(b) ∈ EXPBOLBF
V p: PRED ; s: Seq EXPBF   .
     pred(p,s) ∈ EXPBOLBF  <===>
          # s = Aridad(p)
V o: OPlog ; sb: Seq EXPBOL
     aplicaBOL(o,sb) ∈ EXPBOLBF  <===>
          ( # s = Aridad(o)  )  &
          ( V eb  ∈ rng sb  .
              i
            eb ∈ EXPBOLBF )
              i
```

La relación que determina cuando un Ambiente dado contiene en su dominio a todas las variables de una expresión boleana bien formada será llamada _Contiene _ y se define en el siguiente esquema.
expb

```
_Contiene _ : Ambiente <----> EXPBOLBF
      expb
_____________

V a: Ambiente ; eb: EXPBOLBF    .

     a Contiene   eb     <====>
               expb
          Variables  (eb) C dom a
                     expb
```

Por último definiremos la función Evalúa que nos permitirá efectuar la evaluación de toda expresión boleana bien formada.
expb

```
Evalúa  : Ambiente ----> EXPBOLBF --/--> BOLEANO
     expb
_____________

V b: BOLEANO     .

     Evalúa  (a) (valBOL(b)) = b
          expb

V p: PRED ; s: Seq EXPBF    .

     Evalúa  (a) (pred(p,s)) =
          expb
                          S_pred (p) (s ; Evalúa(a) )

     <===>  V e : rng s     .  a Contiene e
```

```
∀ o: OPlog ; sb: Seq EXPBOLBF    .

      Evalúa  (a) (aplicaBOL(o,sb)) =
           expb
                        S_oplog (o) (sb ; Evalúa  (a) )
                                                    expb
      <===>  ∀ eb : rng sb     .  a Contiene eb
                i                                    i
```

N O T A S

(1) ANSI/SPARC es un grupo de estudio en sistemas manejadores de bases de datos formado en 1972 por el Standars Planning and Requeriments Comittee (SPARC) del American National Standars Comittee (ANSI). Para mayores datos sobre este grupo puede consultarse:

"ANSI/X3/SPARC Study Group on Data Base Management Systems: Interim Report" . FDT (Bulletin of ACM SIGMOD) Vol. 7, No. 2, 1975.

(2) El autor aclara en [DATE 83] que el concepto que él manejaba en su anterior libro [DATE 77] "no hace justicia al concepto de modelo de datos" (pag. 187 [DATE 83]).

(3) Ver para detalles:

Codd, E. F.

"A relational model of data for large shared data backs", CACM 13, No. 6, june 1970.

(4) Más adelante se retoman estos conceptos para definirlos más específicamente de acuerdo al modelo central en esta tesis.

(5) La definición formal de los conceptos de la descripción del modelo relacional puede consultarse en

[CODD 79].

(6) Para mas información puede verse [BUCH 80] pag. 48 y [TSICH 82] pag. 100.

(7) Información detallada sobre este modelo y el jerárquico puede consultarse en [TSICH 82] . Además, en dicha referencia se da bibliografía adicional.

(8) Más información sobre este punto puede encontrarse en:

Tsichiritzis, D. et. al.
"Views on data", The ANSI/SPARC data base model,
North Holland, 1977.

(9) Los interesados en profundizar en I.3.1 pueden remitirse a [BUCH 80], [BUCH 84a] y [DAYA 85] que fueron las principales fuentes consultadas en dicha sección.

(10) La misma observación vale para los puntos que se refieren a los requisitos a cumplir por una BD en ambiente CAD, es decir, se enlistan los que tienen más relevancia en cuanto al modelo conceptual.

(11) La aplicación de bases de datos a CAD puede considerarse como un prototipo de los nuevos problemas que se presentan en la actualidad (e incluso de los que se vislumbran en el futuro). Esto implica que los requisitos a satisfacer por bases de datos en ambiente CAD, pueden considerarse requisitos comunes para la mayoría de las aplicaciones modernas.

(12) En la actualidad, los textos de BD introducen el con-

cepto de objeto como concepto básico para describir, en términos de éste, a todos los modelos de datos, incluyendo a los tradicionales.

(13) Algunos autores (por ejemplo ver [DATE 83], pag.227) no consideran apropiado el término "modelos de datos semánticos" sino que prefieren hablar de "modelado de datos semántico". Además, algunas veces se usa indistintamente el término "modelos extendidos" para hacer referencia a modelos semánticos, ya que se considera que todos los nuevos modelos tienen sus bases en alguno de los tres modelos tradicionales. Sobre la etiqueta "semántico" Codd hace en [CODD 79] la siguiente observación (pag. 398):

"Actualmente la tarea de capturar el significado de los datos nunca termina. Así, la etiqueta "semántico" no debe ser interpretada en ningún sentido absoluto. Por otra parte, los modelos de datos más antiguamente desarrollados (y algunas veces atacados como "sintácticos") no carecían de características semánticas (note por ejemplo los conceptos de dominio, llave y dependencias funcionales)".

(14) Puede afirmarse que la generalización y agregación se encuentran presentes en la mayoría de los modelos de datos. La diferencia es que en los modelos tradicionales , los conceptos no habían sido definidos especificamente y no se hacía referencia a ellos en forma explícita ([TSICH 82], pag. 21).

(15) Las ideas de este modelo fueron presentadas por Codd por vez primera en 1979 en la Australian Comptr. Sci. Conf. efectuado en Tasmania, de ahí el nombre de RM/T que según el autor se refiere al sistema formal que define al modelo relacional extendido que propone.

(16) Se ha afirmado incluso (ETSICH 82], pag. 114) que quizá esta inoveción es la más importante de RM/T pues es la base para el desarrollo de otras extensiones.

Se verá mas adelante que en el modelo especificado en este trabajo se retoma este concepto de identificador.

(17) Codd plantea que los problemas del uso de llaves del usuario son:

- Las llaves controladas por el usuario están sujetas a cambio.
- Dos objetos iguales pueden tener llaves distintas.
- Puede ser necesario registrar información acerca de una entidad ya sea antes de que le sea asignado un valor a la llave o después que haya dejado de tenerlo.

Por otro lado, el autor indica que el concepto de llave no es obsoleto en este modelo ya que el usuario puede necesitarla cuando él requiera tener el control de algunos datos, con la ventaja de que no está obligado a inventar una llave si él no lo

desea.

(18) En [SHIP 81] pag. 143 se menciona el artículo respectivo:
Shibley, E. H. ; Kershberg, L.
"Data architecture and data model considerations",
Proc. AFIPS Nat. Computer Conf., Dallas Tex., June 1977, pp. 85-96 .

(19) Un trabajo pionero en redes semánticas puede verse en:
Quillian, M.R.
"Semantic memory", in Semantic Information Processing, Ed. MIT Press, Cambridge Mass., 1968, pp. 227-270.

(20) Ver la nota (2) de la pag. 142 en [SHIP 81], donde el propio autor considera que DAPLEX es esencialmente un modelo de redes semánticas.

(21) Uno de los primeros trabajos importantes en esta área fue el de Roussopoulos Mylopoulos en 1975, puede consultarse en:
Roussopoulos, N. ; Mylopoulos, J.
"Using semantic networks for data base management",
Proc. 1st Int. Conf. Very Large Data Bases, 1975, pp. 144-172.
También el artículo siguiente, de gran influencia en el medio:
Mylopoulos,J. ; Wong, H. K. T.
"Some features of the TAXIS data model", Proc. 6th

Int. Conf. VLDB, 1980, pp. 399-410.

(22) Aunque en los artículos fundamentales mencionados en este capítulo no aparece como tal el término "modelo de datos orientado a objetos", se ha utilizado en este trabajo pues nos parece que capta la esencia del modelo .

(23) Por ejemplo en [BUCH 85] el autor plantea:
" El modelo de agregación molecular está siendo desarrollado como parte de un proyecto que pretende combinar un lenguaje orientado a objetos, TM [GERZ 85], [BUCH 85a]), con un SMBD a la medida de los requerimientos de CAD para formar un ambiente de programación orientada a objetos completo para CAD ".

(24) El aspecto más estudiado ha sido el de programación orientada a objetos. Acerca de la diversidad de definiciones sobre lenguajes y programación orientados a objetos puede verse [PASC 86] y otros artículos de la revista BYTE de agosto de 1986.

(25) Se ha visto la dificultad pedagógica de conceptualizar adecuadamente lo que es un objeto . Sin embargo, la especificación formal que se presentará más adelante aclarará dicho concepto.

(26) La noción de clase surge con el lenguaje SIMULA, considerado como el primer lenguaje orientado a objetos.

(27) Más adelante en este trabajo se podrá observar cierta analogía entre el concepto de superclase y el de clase de objeto molecular, y entre el concepto de subclase y el de clases de objetos agregados de la molécula.

(28) En [BUCH 85] a estos objetos se les llama "objetos atómicos", a los objetos complejos se les llama "objetos moleculares" o "agregados moleculares" y a los roles se les llama "ligas". Los interesados pueden consultar la parte 5 (Agregación molecular) del texto citado.

(29) La definición de un lenguaje de expresiones boleanas para definir condicionamiento de roles se presenta en el Apéndice II.

(30) En las aplicaciones de BD al diseño ingenieril es necesario poder almacenar instancias iguales de objetos. En momentos posteriores dichas instancias pueden diferir entre sí con el avance de los diseños.

(31) Retomamos el término agregación molecular de [BUCH 84b]. En la definición del modelo orientado a objetos se ha buscado satisfacer la necesidad del manejo de los cuatro distintos tipos de objetos moleculares que en dicho texto se discuten: objetos moleculares disjuntos, no-disjuntos, recursivos y no-recursivos, más adelante se definen estos tipos y se muestran ejemplos.

(32) El Modelo no establece, en principio, roles especiales a los que corresponda automáticamente una forma de herencia predefinida ya que esto sería limitar los alcances y posibilidades del mismo. Sin embargo, en aplicaciones especiales no hay nada que impida que se asocien a ciertos roles, capacidades de herencia particulares y fijas, si así se desea.

(33) En los modelos de datos se distinguen varios tipos de restricciones, sin embargo, nosotros usamos el término para referirnos sólo a condicionamientos de roles.

(34) La cita está tomada de [JONE.80]. El material consultado para la redacción de la presente sección y la III.2 fue principalmente (incluyendo la referencia anterior) : [QUIN 85], [GUTT 82], [LISK 75], Y [NAUR 82].

(35) Este punto y el anterior son evitados puesto que la utilización de un lenguaje de especificación formal garantiza la posibilidad de probar que por ejemplo un programa (escrito a su vez en un lenguaje formal) satisface la especificación, lo cual es imposible de probar si la especificación es informal.

(36) El interesado en estos temas puede consultar [JONE 80a], [LISK 75] y [GUTT 82].

(37) El lenguaje Z en la actualidad sigue

evolucionando y se encuentra en preparación un nuevo manual de Z editado por el Dr. Bernard Sufrin de la Universidad de Oxford.

(38) Note que Atr-Prim = dom AP.

(39) La mayoría de las funciones que definiremos serán parciales y finitas por el mismo tipo de argumentos, por lo cual, en adelante se omitirán anotaciones técnicas al respecto.

(40) La mayoría de los esquemas se irán redefiniendo, por lo que en adelante se omitirán los comentarios al respecto. En general, cualquier referencia a un esquema se referirá a la última definición de él.

(41) En adelante asumiremos esta suposición, a menos que se indique otra redefinición sintáctica del conjunto VALOR.

(42) Si la identificación de instancias se estableciera mediante el concepto de "llave", entonces sí sería necesario pedir que un subconjunto mínimo de atributos estuviera instanciado.

(43) Recordemos que c4 no tiene definidos atributos primitivos.

(44) Para tener una idea exacta del tipo de fórmulas que pueden ser expresiones bién formadas del lenguaje, remitimos al lector al Apéndice II.

(45) La función Evalúa lleva implícita, en su definición

formal, una traducción de símbolos al asociar al símbolo con su operador correspondiente. Por ejemplo, se asocia al símbolo "+" con el operador +. Sin embargo, nosotros omitiremos la diferencia entre símbolo y operador debido al tratamiento informal del lenguaje de expresiones. Consulte el apéndice para la formalización.

(46) En este trabajo hemos especificado una familia de lenguajes para definición de atributos para facilitar el modelado del sistema. Sin embargo, podrían definirse varios lenguajes diferentes en estructura, si así conviniera para la definición de cada categoría de atributo.

(47) Note que en el ejemplo se supone que los atributos generados atr3, atr4 ya tienen un valor. Para el caso de la inicialización de BDM podría suponerse que las instancias de los atributos generados no se insertan por el usuario, sino que se generan por el sistema , entonces, para que I23 pudiera aplicarse como evaluador podría remodelarse BDM suponiendo inicialmente que el sistema asigna un símbolo especial a los atributos generados que indique valor indefinido (como NULO o cualquier otro). Con esto se garantiza que los atributos generados siempre pertenezcan al dominio de Valor-Inst. Así, el sistema actuaría en forma general, en caso de inicialización de BDM y en caso de chequeo de valores. Este tipo de consideraciones relacionadas con

con la implantación no se consideran en el nivel de especificación que estamos desarrollando.

(48) El lector puede remitirse a la descripción informal del modelo, en su apartado II.2.1, donde se introduce intuitivamente el concepto de atributos heredados y se presenta un ejemplo en el que se justifica la necesidad de incluir en los nombres de variables la clase que hereda, el atributo y el rol.

(49) Se recomienda al lector revisar la actualización de I9 e I10 en V.1.2.1 y comprobar que, dado el último esquema de DEF-CLASE, dichos invariantes garantizan que la instanciación de atributos heredados es consistente.

(50) Note que este nuevo invariante es un predicado del esquema BDM y no de DEF-CLASE, como en el caso de I22, para atributos generados, ya que éstos solo pueden depender de atributos de la misma clase. En cambio, en el caso de los atributos heredados la dependencia se da entre atributos de clases diferentes, por lo que el contexto para definir el invariante debe ser más general.

(51) Aunque este mecanismo de herencia a varios niveles pudiera ser ineficiente en una implementación, recordamos al lector que la especificación debe hacer abstracción de este tipo de cuestiones, para poder formalizar lo aspectos fundamentales en el sistema.

(52) Note que no es suficiente pedir que c pertenezca a Clase(v).Agregados ya que esto no garantiza que la agregación sea a través de v.Rol, lo cual es un requisito necesario.

(53) Note que para que Ambiente(var1) esté bién definido, es necesario que (Clase(var1), id^{-1}(var1), Atrib(var1)) pertenezcan al dominio de Valor-Inst.

(54) La nota 47 es válida también para atributos heredados y para I30.

(55) Es conveniente que el lector compare este punto con el punto c) referido a los requisitos de variables de expresiones de atributos heredados (Sec. V.1.3.1) y observe la diferencia en el componente v.Rol.

(56) Se sugiere al lector que compare I33 con I26 para que atienda la diferencia entre variables de expresiones de atributos retribuidos y heredados reforzando la observación de la nota anterior.

(57) Note que no es suficiente pedir que Clase(v) pertenezca a Esq(c).Agregados ya que esto no garantiza que la agregación de Clase(v) sea a través de v.Rol, lo cual es un requisito necesario.

(58) La nota 47 también es válida para atributos retribuidos y para I37.

(59) Note que el siguiente esquema define que Iden-Asoc(c, i) = {id: ID-INST / i C Inst(c).IC(id)} donde se especifica contención y no igualdad ya que

se esté suponiendo que los atributos del parámetro i que el usuario proporcione simpre serán atributos primitivos, ya que el sistema incorpora automáticamente a los atributos de otro tipo.

(60) Cuando dicha instancia es igual a otras de la misma clase , esta operación es en realidad la desconexión de todas las instancias iguales de la otra instancia. De aquí en adelante debe tomarse en cuenta este hecho para todas las operaciones.

(61) Denotaremos que una operación exitosa o total no es a nivel de usuario marcando con un asterisco el nombre de la operación.

(62) En todas las operaciones sobre instancias específicas, el usuario sólo debe hacer referencia a atributos primitivos. Los demás atributos son generados por el sistema.

(63) En los ejemplos siguientes de esta sección supondremos siempre que las gráficas que muestran efectos de operación ilustran sólo roles que satisfacen la condición.

(64) El interesado en profundizar en los conceptos de "especificación denotativa" pueden consultar [GORD 79].

BIBLIOGRAFIA

[ALBA 86]

Albano, A. et. al.

A strongly typed, interactive object-oriented database programming language, Dipartamento di Informatica. Universita di Pisa, Corso Italia, 1986, 15 p.

[ATKI 84]

Atkinson, M. P.; Kulkarni, K. G.

Experimenting with the functional data model, Dept. of Computer Science, University of Edinburgh, Edinburgh England, 27 p.

[BATO 84]

Batory, D. S. ; Buchmann, A. P.

"Molecular objects, abstract data types and data models: A framework", Comunicaciones Técnicas, Serie Naranja: Investigaciones, No. 359, IIMAS-UNAM, 1984, 25 p.

[BATO 84]

Batory, D. S. ; Buchmann, A. P.

"Molecular objects, abstract data types and data models: A framework", Proc. 10th Intl. Conf. on Very Large Data Bases, Singapore, Aug. 1984.

[BUCH 80]

Buchmann, Alejandro

" A methodology for logical design of databases for project engineering", Comunicaciones Técnicas, Serie Naranja: Investigaciones, No. 245, IIMAS-UNAM, 1980, 295 p.

[BUCH 80]

Buchmann, Alejandro

A methodology for logical design of databases for project engineering, PhD Thesis, The University of Texas, Austin Texas, 1980.

[BUCH 84]

Buchmann, Alejandro

"Current trends in CAD databases", Computer-Aided Design, Vol. 16, No. 3, May 1984, p. 123-126.

[BUCH 85]

Buchmann, A. ; Perez de Celis, C.

"An architecture and data model for CAD databases", Comunicacione Técnicas, Serie Naranja: Investigaciones, No. 379, IIMAS-UNAM, 1985, 15 p.

[BUCH 85]

Buchmann, A. ; Perez de Celis, C.

"An architecture and data model for CAD databases", Proc. 11th Intl. Conf. on Very Large Data Bases, Stockholm, Aug. 1985.

[BUCH 85a]

Buchmann, A. ; Gerzso, J. M.

Handling heterogeneously formatted data in an object oriented database enviroment, National Computer Graphics Association, Dallas Texas, April 1985.

[BUCH 86]

Buchmann, Alejandro et. al.

"A generalized constraint exception handler for an object-oriented CAD-DBMS", Comunicaciones Técnicas, Serie Naranja: Investigaciones, No. 423, IIMAS-UNAM, 1985, 19 p. (por publicarse en el 1st. Intl. Workshop on Object-Oriented DB Systems, Pacific Grove, Cal., Sept. 1986).

[CARD 85]

Cardelli, Luca ; Wegner Peter

"On understanding types, data abstraction, and polymorphism", ACM Computing Surveys, Vol. 17, No. 4, December, 1985, pp. 471-522.

[CODD 79]

Codd, E. F.

"Extending the database relational model to capture more meaning", ACM Transactions on Database Systems, Vol. 4, No. 4, December 1979, p. 397-434.

[DATE 77]

Date, C. J.

An Introduction to database systems, Addison Wesley The Systems Programming Series, 2a. Ed., 1977, 536 p.

[DATE 83]

Date, C. J.

An introduccion to database systems. Vol II, Addison Wesley, The Systems Programming Series, 1983, 383 p.

[DAYA 85]

Dayal, U.; Buchmann, A.; et. al.

PROBE- A research proyect in knowledge-oriented database systems: Preliminary analysis, Computer Corporation of America, Four Cambridge Center, July 1985, 81 p.

[GERZ 85]

Gerzso, J. M. ; Buchmann, A.

"TM - An object-oriented language for CAD and required database capabilities", in Languages for automation, S. K. Chang (ed), Plenum Press, 1985.

[GORD 79]

Gordon, Michael J. C.

The denotational description of programming languages, Springer-Verlag, 1979.

[GUTT 82]

Guttag, John et. al.

Some remarks on putting formal specifications to productive use, working paper draft -circulated for comment only, PARC/CSL & MIT/LCS, february, 1982, 10 p.

[HAYE 86]

Hayes, Ian et. al.

Specification Case Studies, Oxford University.
Programming Research Group, McGrow-Hill (próxima publicación), 1986, 302 p.

[JONE 80]

Jones, C.B.

"The role of formal specifications in software development", in UK-state of the art report Life-cycle management, Infotech International Limited, Maidenhead, 1980.

[LAUS 86]

Lausen, G.; Schek, J.

Semantic specification of complex objects, Fachbereich Informatik TH Darmstadt, West-Germany, April 1986, 24 p.

[LISK 75]

Liskov, Barbara; Zilles, S.

"Specification techniques for data abstractions", IEEE Transactions on Software Engineering, Vol. SE-1, No. 1, March 1975, pp. 7-18.

[NAUR 82]

Naur, Peter

"Formalization in program development", BIT 22, 1982, pp. 437-453.

[PASC 86]

Pascoe, G.

"Elements of object-oriented programming", BYTE,

August 1986, pp. 139-144.

[QUIN 86]

Quintanilla, Gloria et. al.

"Diseño del intérprete de un lenguaje de programación a partir de su especificación formal", Comunicaciones Técnicas, Serie Amarilla: Desarrollo, No. 51, IIMAS-UNAM, 1986, 65 p.

[QUIN 85]

Quintanilla, Gloria .

"Case studies in specification: The matematical structure relation", Comunicaciones Técnicas, Serie Naranja: Investigaciones, No. 391, IIMAS-UNAM, 1985, 75 p.

[RENT 82]

Rentsch, Tim

"Object oriented programming", SIGPLAN Notices, 1982, pp. 51-57.

[SHIP 81]

Shipman, David

"The functional data model and the data language DAPLEX", ACM Transactions on Database Systems, Vol. 6, No. 1, March 1981, p. 140-173.

[SMIT 77]

Smith, J.M. ; Smith, D. C. P.

"Database abstractions: aggregation and generalization", ACM Transactions on Database Systems, Vol. 2, No. 2,

June 1977, pp. 105-133.

[TSICH 82]

Tschiritzis, Dionysios; Lochovsky, Federick

Data Models, Prentice-Hall. Sftware Series, 1982,
381 p.