CONSTRUCCIÓN AUTOMÁTICA DE PROTOTIPOS DE SISTEMAS DE INFORMACIÓN A PARTIR DE MODELOS CONCEPTUALES EN DADES.

Sistac, J., Olivé, A.

Facultad de Informática. Jordi Girona Salgado, 31.
08034 Barcelona.

Resumen. Muchos esfuerzos actuales de investigación y desarrollo en el área de los Sistemas de Información (SI) se dirigen hacia la construcción automática de prototipos. Tal construcción parte de una definición previa del SI en un cierto modelo. La mayor parte de las experiencias realizadas hasta la fecha se basan en modelos lógicos o conceptuales operacionales. Nuestro trabajo explora una vía alternativa, consistente en partir de un modelo conceptual declarativo, utilizando en particular el lenguaje DADES. Hemos construído un generador que produce un sistema prototipo equivalente al definido en el modelo DADES. En este artículo describimos las características principales del lenguaje y del generador.

INTRODUCCIÓN

Muchos esfuerzos actuales de investigación y desarrollo, en el área del diseño y construcción de sistemas de información (SI), se dirigen hacia la construcción de prototipos de estos sistemas. Tales esfuerzos están plenamente justificados por la existencia de diversos intereses prácticos (y teóricos) en los logros que puedan alcanzarse.

Seguramente, el interés principal lo constituye la necesidad de verificar la adecuación de los requerimientos de un SI, o de sus especificaciones, a las necesidades reales de los usuarios (Naumann et al., 1982). Tal verificación no puede ser formal, ni incluso en el caso que las especificaciones se escriban en un lenguaje formal, porque no es posible representar formalmente lo que los usuarios realmente necesitan o desean. En consecuencia, el diseñador debe utilizar procedimientos informales de verificación, consistentes en interpretar, explicar, describir, etc., a los usuarios el significado exacto de los requerimientos, confiando en que estos usuarios podrán formarse un juicio sobre la adecuación entre lo que desean y lo que van a obtener cuando el sistema esté construído. Tal confianza es ilusoria cuando los usuarios no saben o no pueden saber lo que desean, o cuando no pueden formarse una idea exacta de lo que implican unos ciertos requerimientos.

Una vía alternativa de verificación la constituye la construcción de un prototipo a partir de los requerimientos, entendiendo por prototipo "una versión inicial de un SI que exhibe las características esenciales del sistema final" (Alaví, 1984). Si se dispone de un prototipo, los usuarios pueden experimentar con él, descubrir sus características y detectar aquellos aspectos que no concuerdan con sus deseos o necesidades. Se anticipa el resultado final para verificar que éste será el correcto. No hay ninguna duda que esta vía alternativa es mucho más conveniente que la anterior, lo que justifica que, si los prototipos pueden construirse a un costo razonable, se recomiende fuertemente su construcción (Berrisford et al., 1979), (Davis, 1982).

La forma usual de construcción de prototipos consiste en emplear herramientas específicas y lenguajes de muy alto nivel, que permiten disponer de un prototipo con un esfuerzo reducido. Normalmente, estos prototipos no son completos, pero tienen las características más importantes del sistema final. Ejemplos de este enfoque son ACT/1, Mason et al. (1983), y USE, (Wasserman, 1982).

Otro enfoque posible, hacia el que se dirigen muchas líneas de

investigación, es el de generar automáticamente el prototipo a partir de un modelo del SI, que debe estar definido mediante un lenguaje formal. El prototipo tiene ahora exactamente las mismas características que el sistema final, pero con una eficiencia menor (tiempo de respuesta, consumo de recursos, etc.). Naturalmente, si se dispone de un generador de este tipo, el esfuerzo de obtención del prototipo es nulo. Nuestro trabajo se inscribe dentro de este segundo enfoque.

A su vez, los generadores de prototipos pueden clasificarse en función del tipo de modelo de partida. Hasta hace poco tiempo, el tipo de modelo usual era el lógico, en el que un SI se define en términos de las operaciones lógicas que debe realizar y de sus relaciones, vía flujos o archivos lógicos de información. Ejemplos de esta clase de generadores son el del SREM, Bell et al. (1977), que parte de modelos definidos en el lenguaje RSL, o los basados en PSL (Teichroew et al., 1977) como el descrito en McCoyd et al., (1982).

Otra posibilidad consiste en partir de un modelo de tipo conceptual. Estos modelos se basan en los conceptos "Universo de discurso" y "Universo de información". El Universo de discurso consiste en todas aquellas entidades de interés del mundo real que han existido, existen o existirán (ISO, 1982). El Universo de información, en cambio, consiste en todas las informaciones de interés sobre las entidades del Universo de discurso. El estado del Universo de discurso en un instante determinado consiste en todas aquellas entidades que existen en aquel instante. A su vez, el estado del Universo de información en un instante determinado consiste en todas las informaciones de interés existentes hasta aquel instante.

Un modelo conceptual consiste en la definición de los estados posibles del Universo de información y de las transiciones posibles de estado. Las entradas al SI se definen entonces en términos de la información que añaden al Universo de información, y las salidas en términos del subconjunto de información que extraen del mismo.

Dentro de los modelos conceptuales se distinguen dos enfoques: operacional y declarativo. El primero es el que adoptan la mayor parte de los lenguajes conceptuales existentes, tales como el ACM/PCM (Brodie et al., 1982), o el TAXIS (Mylopoulos et al., 1980). Se han desarrollado, o están en vías de desarrollo, generadores de prototipos para muchos de éllos. Un ejemplo puede ser el del GIST, (Balzer et al., 1982) y (Feather, 1982).

Nuestro trabajo se basa, en cambio, en modelos coneptuales declarativos, para los que no existen aun experiencias (conocidas) de generación de prototipos. Hemos utilitzado como lenguaje de partida el DADES, (Olivé, 1982), aunque también se podría aplicar, con ligeros retoques, a otros lenguajes como el CIAM (Gustaffson el al., 1982).

En la Sección siguiente se resumen las características principales de los modelos conceptuales declarativos. A continuación, en la Sección 3 se describen los elementos principales del lenguaje DADES. Ello constituye la base para, en la Sección 4, exponer el enfoque adoptado para la generación de prototipos y la arquitectura del sistema. El sistema está aun en fase de experimentación, susceptible de ampliarse y mejorarse en diversas líneas, que se apuntan en las conclusiones.

MODELOS CONCEPTUALES. ENFOQUE DECLARATIVO.

En esta sección trataremos de caracterizar los modelos conceptuales declarativos, independientemente del lenguaje en que se definen.

En este tipo de modelos, el concepto "tiempo" juega un papel fundamental. Toda información $\underline{i}$ del Universo de información tiene un instante asociado, T(i), que indica el instante (de observación o de ocurrencia) al que se refiere la información. Si la información se refiere a un intervalo, entonces aquel instante es el último instante del intervalo. Ejemplos:

T(Juan nació el 10/3/1985) =
= 10/3/1985
T(Durante el 1985 nacieron 100 personas) = 31/12/85.
T(La previsión de ventas para el 1985, hecha el 10/4/84, es de x pesetas) = 10/4/84.

Estos instantes deben expresarse en una unidad de tiempo suficientemente fina como para evitar ambigüedades. Tal unidad de tiempo (segundo, día, año, etc) es única para todo el sistema.

Cada SI tiene un período de vida, comprendido entre un instante inicial de funcionamiento t_o y un instante final t_f. El período de vida se conceptualiza como un conjunto de instantes $T=\{t_o,t_1,\ldots,t_f\}$ comprendido entre t_o y t_f, donde cada instante se expresa en la unidad de tiempo del sistema.

El Universo de información en un instante $t \in T$, UI(t) contiene todas aquellas informaciones cuyo instante es menor o igual a t: $UI(t) = \{i/T(i) \leqslant t\}$. A medida que transcurre el tiempo, UI(t) va conteniendo más y más informaciones.

En particular, el Universo de información contiene informaciones sobre los acontecimientos que se producen en el Universo de discurso. Cada acontecimiento da lugar a una o más informaciones. De hecho, las entradas al SI son exclusivamente informaciones sobre acontecimientos.

Las informaciones del Universo de información deben modelarse mediante algún "modelo de datos". Los tipos de información resultantes se definen en el Esquema conceptual. El esquema debe contener también la definición de las restricciones estáticas y dinámicas de las informaciones. Algunas de estas restricciones ya están incorporadas (son implícitas) en las construcciones básicas del modelo de datos, mientras que otras deben definirse explícitamente.

El conjunto de informaciones de UI(t) se divide en dos subconjuntos: El núcleo, N(t), y la parte derivada, D(t). El núcleo contiene todas aquellas informaciones que se refieren a los acontecimientos producidos en el Universo de discurso. La parte derivada, en cambio, contiene aquellas informaciones que pueden deducirse del núcleo, del Esquema Conceptual y/o de otras informaciones derivadas. La derivación se efectúa mediante reglas de derivación.

El SI produce una salida cada vez que un acontecimiento externo del entorno lo solicita, o en condiciones predefinidas. Esta información se extrae (es un subconjunto) del Universo de información existente en el instante de producción de la salida. En un mismo instante pueden producirse diversas salidas.

EL LENGUAJE DADES

DADES es uno de los lenguajes que pueden emplearse para definir un modelo conceptual declarativo de un SI. Un modelo en DADES consta de tres partes:

- Definición Esquema conceptual
- Definición Entradas
- Definición Salidas

En esta sección exponemos las características principales del lenguaje, que se ilustran mediante un ejemplo sencillo de gestión de un almacén.

Definición del Esquema

En DADES se utiliza el modelo relacional para modelar el Universo de información y el álgebra relacional para definir subconjuntos de este Universo. En este modelo, los tipos de información se definen mediante esquemas de relación. Cada esquema de relación consta de un nombre, los atributos de la relación, el dominio de cada uno de éllos y los atributos clave.

El esquema conceptual consiste entonces en la definición de:

- Los esquemas de relación
- Los dominios
- Las restricciones
- Las reglas de derivación

En el ejemplo, podrían existir los cuatro esquemas de relación siguientes:

PRODUCTOS (PRODUCTO, NOMBRE, PRECIO)
Indica los productos existentes en el almacén, sus nombres y precio unitario. Se supone que estas informaciones son invariables durante el período de vida del sistema.

VARSTOCK (PRODUCTO, FECHA, CANTIDAD)
Indica la variación de stock de un producto en una cierta fecha.

STOCK (PRODUCTO, FECHA, CANTIDAD)
Indica la cantidad en stock de un producto en una cierta fecha.

VALOR-ALMACEN (FECHA, VALOR)
El valor del almacén en una cierta fecha es igual a la suma de los valores de los stocks de todos los productos.

Omitiremos aquí la definición de los dominios y de las restricciones. Las reglas de derivación se describirán en un apartado posterior.

Definimos como unidad de tiempo del sistema, un día (día/mes/año), con lo que DADES asumirá que en el esquema están incluídas algunas relaciones de tiempo, tales como:

INSTANTES (INSTANTE: FECHA)
Indica los instantes (fechas) existentes en el período de vida.

INSTANTE-ANTERIOR (INSTANTE: FECHA, ANTERIOR: FECHA).
Indica para cada instante del período de vida el instante anterior en el mismo (los instantes no son necesariamente correlativos).

Estas relaciones pueden ser utilizadas libremente por el diseñador en otras definiciones del modelo.

Definición de las entradas

En DADES, las entradas al SI pueden tener dos orígenes: (1) Como consecuencia de un acontecimiento externo; y (2) Como consecuencia de una petición formulada por el SI. En este trabajo consideraremos sólo el primer caso.

La definición de una entrada consiste en la definición del acontecimiento que la produce y la definición del contenido de información que aporta. Este contenido se define mediante una expresión del álgebra relacional. En el ejemplo, hay dos acontecimientos relevantes. El primero es una variación de stock de un cierto producto. Se definiría así:

event VARIACIÓN-DE-STOCK (PRODUCTO, FECHA).
Este acontecimiento se produce cada vez que hay una variación del stock de un cierto producto. El acontecimiento se identifica por el producto afectado y la fecha de variación.

La información que se recoge de este acontecimiento (entrada al SI) se definiría:
input I1;
for all PRODUCTO p FECHA f in VARIACIÓN-DE-STOCK;
provides VARSTOCK (PRODUCTO = p, FECHA = f).
indicando que hay una entrada de tipo I1 por cada ocurrencia del tipo de acontecimiento VARIACIÓN-DE-STOCK y que la información que aporta es un tuplo de la relación VARSTOCK.

El segundo acontecimiento de interés es la inicialización del sistema. Se definiría como sigue:
event INICIALIZACIÓN.
Se produce al inicio del período de vida del sistema (una sola vez).
y la información que aporta se definiría:
input I2;
for all INICIALIZACIÓN;
provides PRODUCTOS.
indicando que se entra de una sola vez el código, el nombre y el precio de todos los productos.

Reglas de derivación

Los tipos de información definidos en el esquema se clasifican en básicos y derivados. Los primeros son aquellos cuyas informaciones provienen directamente del exterior (entradas) y derivados todos los restantes. Por cada tipo de información derivado hay que especificar una o más reglas de derivación. En el ejemplo, son derivados los esquemas de relación STOCK y VALOR-ALMACÉN.

Una regla de derivación define cómo se deriva una ocurrencia concreta de una información de un cierto tipo. Consta de las partes siguientes:

- Intensión, que define el tipo de información que se deriva.

- Dominio, que define la extensión del tipo de información para la que está definida la regla. Puede ser total o parcial. Si es parcial, la regla deberá complementarse con otras tales que, entre todas ellas, definan la totalidad de la extensión.

- Las informaciones origen, a partir de las cuales la regla de derivación deriva la nueva información.

- La función, que aplicada a las informaciones origen da la información deseada.

La regla correspondiente a STOCK podría definirse como sigue:
derivation rule DR1;
for all PRODUCTO p in PRODUCTOS;
for all INSTANTE f in INSTANTES;
STOCK(PRODUCTO=p,FECHA=f) =
= F1(<p,f>,
VARSTOCK(PRODUCTO=p,FECHA=f),
STOCK (PRODUCTO=p,
INSTANTE=previo(f))).
que significa que la intensión es el tipo STOCK; el dominio es cada pareja <p,f> ∈ PRODUCTOS [PRODUCTO] x INSTANTES (el stock está definido por cada producto y fecha); las informaciones origen son tres:
- La pareja <p,f>
- Las variaciones de stock de p en f.
- El stock de p en la fecha previa de f. Previo (f) es una función temporal basada en la relación INSTANTE-ANTERIOR.
y F1 es el nombre de la función que, aplicada a las informaciones origen, obtiene STOCK(PRODUCTO=p,

FECHA=f). Hasta el momento, DADES no tiene un lenguaje particular para la definición de estas funciones, por lo que debe utilizarse cualquier lenguaje que resulte conveniente, que en algunos contextos puede ser el lenguaje natural. En este ejemplo, F1 debería realizar la suma algebraica del stock anterior y de la variación del stock.

Por su parte, la regla correspondiente a VALOR-ALMACÉN podría definirse:

```
derivation rule DR2;
  for all INSTANTE f in INSTANTES;
    VALOR-ALMACEN (FECHA=f) =
      = F2(<f>,
        STOCK (FECHA =f),
        PRODUCTOS [PRODUCTO,PRECIO] ).
```

indicando que el valor del almacén en una fecha f se obtiene, mediante F2, a partir de f, de los stocks en f de todos los productos y de todos los precios de los productos.

Definición de las salidas

Al igual que las entradas, en DADES una salida puede tener dos orígenes: (1) como consecuencia de un acontecimiento externo que la solicita; y (2) cuando se satisfacen unas condiciones definidas previamente (acontecimiento interno). En este trabajo consideraremos sólo este último caso.

La definición de una salida consistirá entonces en la definición de la condición de producción y en la definición del contenido de información de la misma. En el ejemplo, una de las salidas deseadas consiste en el valor del almacén, que debe producirse diariamente. Su definición sería:

```
output O1;
  for all INSTANTE f in INSTANTES;
    provides VALOR-ALMACEN (FECHA=f),
```

donde "for all" indica la condición de producción (por cada fecha del período de vida) y "provides" indica, en álgebra relacional, la información deseada.

Cualquier expresión del álgebra relacional puede emplearse en ambos sitios, lo que proporciona una potencia de definición considerable. Sirva como ejemplo la siguiente definición:

```
output O2;
  for all INSTANTE f in INSTANTES;
    for all PRODUCTO p in PRODUCTOS
      [PRODUCTO]-VARSTOCK(FECHA=f)
        [PRODUCTO];
          provides STOCK(PRODUCTO=p,
            INSTANTE=f)*PRODUCTOS.
```

que indica que, por cada fecha f y producto p que no ha tenido variación de stock en f, hay que obtener el código del producto p, su nombre, su precio y la cantidad en stock en f.

GENERACIÓN DE PROTOTIPOS

Arquitectura del sistema

El sistema consta de 3 componentes: analizador, preparador y generador, tal como se indica en la figura 1.

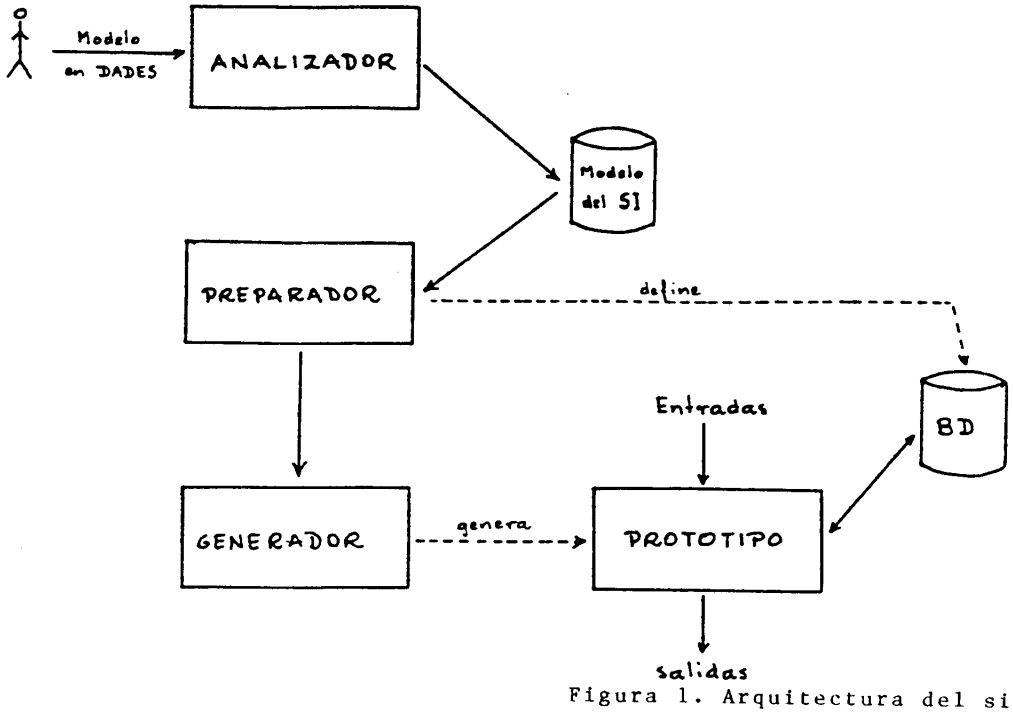

Figura 1. Arquitectura del sistema.

El analizador tiene por función recibir la definición de un modelo conceptual de un sistema de información, escrito en el lenguaje DADES, almacenarla en una Base de Datos, verificarla y proporcionar cuanta información sea requerida por el diseñador. Significa pues que el analizador debe ser capaz de realizar el análisis léxico, sintáctico y semántico del texto que define el modelo conceptual del sistema de información de que se trate, haciendo un tratamiento completo de errores e inconsistencias, y almacenando el texto ya verificado, en una forma conveniente para que pueda ser utilizado por el preparador y el generador en etapas sucesivas.

No hay unicidad, evidentemente, sobre la forma conveniente de almacenamiento, pero si que estan perfectamente definidas las funciones del analizador, y como veremos enseguida tambien las del preparador. para determinar cual podria ser la forma conveniente de almacenamiento, resolvimos empezar a construir el sistema a partir del generador de prototipos, y que fuera el generador quien marcara las exigencias, dada su complejidad.

El preparador, a partir de la información almacenada, determina y define la estructura necesaria de la Base de Datos y determina también la secuencia de operaciones a realizar por el prototipo generado, concretamente el orden de secuencia en que entradas, salidas y reglas de derivación deben producirse en una unidad de tiempo, satisfaciendo las necesarias precedencias de información. También aquí, aunque está clara la función del preparador, debían tomarse algunas decisiones importantes como el SGBD a usar, la forma de almacenamiento de la secuencia de operaciones del prototipo, así como la optimización de la secuencia pensando en transformar ciertas serializaciones en ejecuciones en paralelo.

El generador es un programa escrito en Pascal, que genera a su vez otro programa, también escrito en Pascal, que será el prototipo del Sistema de Información, que en una unidad de tiempo ejecutará las entradas, las salidas y las reglas de derivación del sistema según la secuencia marcada por el preparador. El generador, pues, recibe la información facilitada por el analizador y el preparador, y con ellas genera un prototipo que interaccionará con la base de datos definida previamente por el preparador.

Un problema importante, objeto de decisión previa, fue el de utilizar un programa generador en lugar de utilizar un programa genérico, general para todos los prototipos, debidamente parametrizado. Aunque la segunda opción aparecía en un primer momento como más simplificada y quizá más elegante en que una misma estructura se conservaba variando simplemente los parámetros, apareció enseguida que debían también modificarse no sólo las condiciones de los bucles sino también el número de bucles, es decir, que en el fondo debía modificarse la misma estructura y además en el momento de la ejecución debían ser pasados estos parámetros con innumerables llamadas (calls), lo cual disminuiría notablemente la eficiencia del prototipo. Mucho mejor era ocupar más tiempo en generar el prototipo con los parámetros ya sustituídos, lo que permitiría en las ejecuciones subsiguientes una mayor rapidez, aunque el programa generador pudiera resultar algo más complicado. De hecho, la experiencia ha demostrado que con un programa generador, aunque quizá complejo de escribir, existe mayor ductilidad para modelar el prototipo.

Entorno

Todo el sistema se ejecuta sobre un VAX 11/785 utilizando como base de datos el sistema relacional Rdb de Digital. Nos decidimos por el Rdb por su facil integración con el lenguaje "host" Pascal, lenguaje de programación en el que se ha desarrollado todo el proyecto y también dada la popularidad del VAX entre las universidades.

En la fase actual de implementación, los análisis proporcionados por el analizador y el preparador se guardan en tablas (arrays of records), aunque está decidido que en futuras versiones se almacenarán en la misma base de datos relacional usada por el prototipo.

Se han necesitado unas 10 tablas, que actualmente almacenan información sobre la secuencia del preparador, el nombre de todos los atributos y sus dominios, el nombre de todos los esquemas de relación con sus atributos, sean éstos clave o no, la relación de todas las entradas, salidas y reglas de derivación, con detalle de si las desencadenan acontecimientos externos o internos al SI, número de parámetros, etc. Para representar los atributos

correspondientes a una relación y también para las relaciones que constituyen una expresión relacional y otros casos similares, se ha utilizado la estructura Pascal del "set".

Funcionamiento del Prototipo

El prototipo funciona a base de un gran bucle que se repite por cada instante del periodo de vida del sistema. En cada ciclo se ejecutan las distintas entradas, salidas y reglas de derivación según el orden establecido por el preparador. En principio hay tres grandes tipos de estructuras preprogramadas, correspondientes a las entradas, salidas y reglas de derivación, que generan un trozo de programa correspondiente a cada una de ellas.

Las entradas pueden tener dos orígenes: (1) como consecuencia de un acontecimiento externo y (2) como consecuencia de una petición formulada por el SI. En el primer caso aparece por pantalla el tipo de acontecimiento externo que lo provoca, e inmediatamente pueden introducirse los valores de los diversos atributos identificadores del acontecimiento. Hasta completar la información proporcionada por la entrada, faltarán en general aún muchos atributos a los que asignar valores, pero dado que las relaciones de entrada pueden ser múltiples, y que muchos atributos pueden ser comunes a varias relaciones y que algunos de ellos son clave y otros no, el orden en que dichos atributos han de ser introducidos no es en absoluto indiferente, debiéndose establecer un arbol de precedencias entre ellos, de forma que se requieran por pantalla en el orden adecuado. Este arbol lo construye el propio generador con métodos de combinatoria adecuados.

En el segundo tipo de entradas, consecuencia de una petición formulada por el propio SI, el tratamiento de los atributos especificados en el acontecimiento interno es parecido al que se explicará en las salidas. Los demás atributos hasta completar todos los de la entrada, se tratan como los de las entradas ya explicadas, construyendo el adecuado arbol de atributos al que deberemos asignar valor.

Las salidas, al igual que las entradas, pueden tener dos orígenes: (1) como consecuencia de un acontecimiento externo que las solicita y (2) cuando se satisfagan unas condiciones definidas previamente (acontecimiento interno). En ambos casos, así como en las reglas de derivación, la mayor dificultad de implementación reside en la interpretación del acontecimiento interno, que como ya explicamos, implica condiciones de salida o producción para ciertos atributos a lo largo del periodo de vida del sistema. Eso implica que para una unidad de tiempo deberá ejecutarse una salida o una regla de derivación sólo para las tuplas que no las hubieran efectuado aún desde el principio del periodo de vida del sistema, y teniendo presente que el acontecimiento interno se refiere a la totalidad del ciclo de vida del sistema. Un primer procedimiento para resolver el problema, podría consistir en un marcaje de las tuplas que ya se han tratado, pero al ser múltiples los tratamientos a los que se somete una tupla, también deberían ser múltiples los marcajes, y además en número variable.

La solución adoptada se basa en añadir un atributo de tiempo T a todos los esquemas de relación del núcleo, transparente al usuario aunque no al prototipo, que deberá marcar cada tupla con el valor del tiempo en que dicha tupla se ha insertado en la Base de Datos. Sea T(r) el tiempo en que la tupla r se ha añadido a la Base de Datos. Es evidente que T(r), para una tupla r correspondiente a una entrada, es el instante en que se produce dicha entrada, y si se trata de una regla de derivación será el instante en que dicha tupla se haya calculado.

Sean R/t las tuplas de la relación R añadidas en el instante de tiempo t. Es facil descubrir que si R es un esquema de relación del núcleo,

$$R/t = \{r/r \in R \wedge T(r) = t\}.$$

También es facil deducir para la operación relacional de selección que:

$$R(cond)/t=\{r/r\in R\wedge cond(r)=true\wedge T(r)=t\}$$

que es fácilmente implementable con la ayuda del atributo de tiempo, según la siguiente expresión:

$$R(cond)/t = R(cond \wedge T = t)$$

Para la operación proyección, las tuplas añadidas a la proyección, por ejemplo sobre dos atributos A,B de una relación R, en el tiempo t, sería:

$$R[A,B]/t = (R/t)[A,B] - (R/<t)[A,B]$$

que debidamente transformada puede implementarse en el prototipo como:

$$R[A,B]/t = R(T=t)[A,B] - R(T<t)[A,B]$$

Parecidas transformaciones como las

aplicadas a la selección y a la proyección, han sido estudiadas para las demás operaciones del algebra relacional.

Una vez determinados los acontecimientos internos que se han producido en un cierto instante, es ya muy facil ejecutar las salidas, sea por pantalla, impresora u otros dispositivos, teniendo en cuenta que las expresiones relacionales son facilmente expresables con un sólo "rse" (record selection expression) con el sistema relacional ya mencionado.

Las salidas del tipo (1) serán tratadas exactamente como las del tipo (2), con la única diferencia de que el acontecimiento externo será como una entrada por pantalla.

Por último, las reglas de derivación se tratan, en cuanto al dominio, del mismo modo que las salidas. En la versión actual, la función se implementa como un "procedure" Pascal que debe definir el propio diseñador. El programa prototipo "llama" a este procedimiento, transmitiéndole los parámetros correspondientes a las informaciones origen. El procedimiento debe calcular la información derivada, que el programa prototipo almacenará en la Base de Datos. El tratamiento de las expresiones relacionales de las informaciones origen es muy similar al tratamiento de las salidas.

Sólo nos resta advertir, que la escritura de dichos procedimientos, que serán escritos por el propio diseñador, debería ser facilitada por parte del sistema que estamos explicando. Precisamente por eso, es el propio generador quien escribe el encabezamiento del procedimiento, así como las declaraciones de tipos y variables, debiendo escribir el diseñador únicamente el cuerpo del procedimiento.

Para facilitar aún más el trabajo del diseñador, en dicho cuerpo del procedimiento no existirán jamás operaciones de lectura o escritura de la Base de Datos. La lectura de las tuplas necesarias, la realiza el prototipo antes de la "llamada" al procedimiento, almacenándolas en 'records' o 'arrays de records', que serán precisamente los parámetros antes citados. Ejecutado ya el procedimiento, será el prototipo quien grabará la(s) tupla(s) resultante(s) en la Base de Datos.

CONCLUSIONES

El sistema descrito está aún en fase de implementación, aunque ya se han realizado experimentaciones parciales que permiten asegurar su viabilidad técnica. Con ello esperamos demostrar que la construcción automática de prototipos a partir de modelos conceptuales declarativos es también posible.

No obstante, creemos que el sistema debería mejorarse en diversos sentidos. Probablemente, los más importantes son:

1) En la versión actual, el sistema almacena en la Base de Datos todas las informaciones del universo de información, tanto las provenientes de las entradas como las derivadas. Las informaciones no se eliminan nunca, con lo que el tamaño de la Base de Datos crece indefinidamente, a pesar de que muchas de estas informaciones no se utilizarán posteriormente. Una mejora evidente consistiría en desarrollar un algoritmo que determinase el instante a partir del cual una información deja de ser util, con lo que podría suprimirse a partir de este instante.

2) A su vez, en el sistema actual el preparador obtiene una secuenciación de acciones a realizar que no está optimizada, en el sentido que diversas acciones podrían agruparse para reducir el consumo de recursos. Por ejemplo, la obtención de una información derivada podría ser simultánea con la producción de una salida. Debería, pues, desarrollarse un procedimiento de optimización.

RECONOCIMIENTOS

El presente trabajo ha contado con una ayuda de la Comisión Asesora de Investigación Científica y Técnica (3437-83).

REFERENCIAS

Alavi, M., (Junio 1984)., "An assessment of the prototyping approach to information systems development". Comm. ACM, Vol. 27, No. 6, pp. 556-563.

Balzer, R.M.; Goldman, N.M.; Wile, D.S. (1982), "Operational specification as the basis for rapid prototyping". En Squires et al., pp. 3-16.

Bell, T.E.; Bixler, D.C.; Dyer, M.E. (Enero 1977), "An extendable approach to computer-aided software requirements engineering". IEEE Trans. on Software Eng., Vol. SE-3, No. 1, pp. 49-59.

Berrisford, T.R.; Wetherbe, J.C., (Marzo 1979), "Heuristic development: A redesign of systems design", Management Inf. Sys. Quarterly, Vol. 3, No. 1, pp. 11-19.

Brodie, M.L.; Silva, E. (1982), "Active and passive component modelling: ACM/PCM", En Olle et al., pp. 41-91.

Davis, G.B., (1982), "Strategies for information requirements determination", IBM Systems Journal, Vol. 21, No. 1, pp. 4-30.

Feather, M.S. (1982), "Mapping for rapid prototyping". En Squires et al., pp. 17-24.

Gustaffson, M.R.; Karlsson, T.; Bubenko, J.A., (1982), "A declarative approach to conceptual information modelling", En Olle et al., pp. 93-142.

ISO (1982), "Concepts and terminology for the conceptual schema and the information base", ISO/TC17/SC5, N. 695, Editado por J.J. van Griethuysen.

Mason, R.E.A.; Carey, T.T., (Mayo 1983), "Prototyping interactive information systems", Comm. ACM, Vol. 26, No. 5, pp. 347-354.

McCoyd, G.G.; Mitchell, J.R. (1982), "System sketching: The generation of rapid prototypes for transaction based systems", En Squires et al., pp. 127-132.

Mylopoulos, J.; Bernstein, P.A.; Wong, H.K.T. (Junio 1980), "A language facility for designing database-intensive applications", ACM TODS, Vol. 5, No. 2, pp. 185-207.

Naumann, J.D.; Jenkins, A.M., (Septiembre 1982), "Prototyping: The news paradigm for systems development". Management Inf. Sys. Quarterly, Vol. 6, No. 3, pp. 29-44.

Olivé, A. (1982), "DADES: A methodology for specification and design of information systems", En Olle et al., pp. 285-334.

Olle, T.W.; Sol, H.G.; Verrijn-Stuart, A.A. (Eds.), (1982), "Information systems design methodologies: a comparative review", North-Holland.

Squires, S.L.; Branstad, M.; Zelkowitz, M. (Eds)., (Diciembre 1982). "Working papers from the ACM SIGSOFT. Rapid prototyping workshop", ACM Softw. Eng. Notes, Vol. 7, No. 5.

Teichroew, D.; Hershey, E.A. (Enero 1977), "PSL/PSA: A computer-aided technique for structured documentation and analysis of information processing systems", IEEE Trans. on Software Eng., Vol. SE-3, No. 1, pp. 41-48.

Wasserman, A.I. (1982). "The user software engineering methodology: an overview", En Olle et al., pp. 591-628.

SUMMARY

Many research efforts in the Information Systems (IS) area aim at building prototypes. These prototypes are produced by generators. Input to such generators consist of a formal model of the IS to be built. Most of the existing proposals are based on logical models or conceptual models with an operational approach. Our work tries to explore a new way, consisting on using a conceptual model with a declarative approach. In particular we use the DADES language. We have built a generator which produces a prototype system equivalent to the system defined in DADES. This paper describes the main features of both the language and the generator.