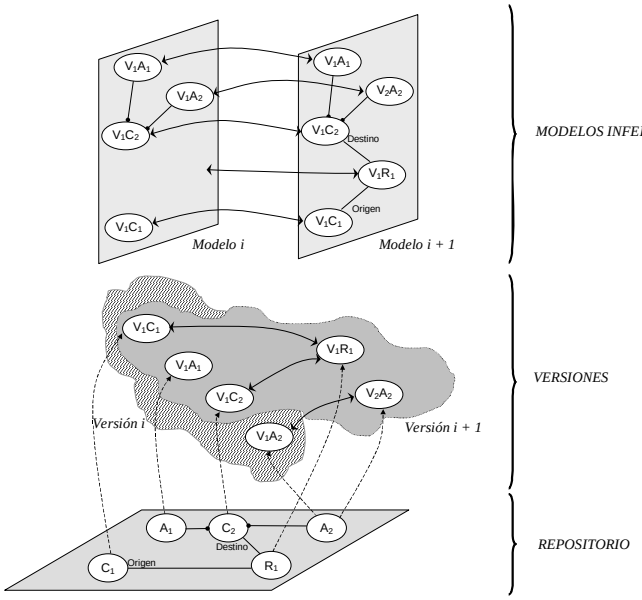
Administración de Versiones de Modelos en una Herramienta de Soporte para Análisis y Diseño Orientado a Objetos

S. Gonnet[*+], R. Holzer[*], H. Melgratti[*] y H. Leone[*+]
[*]GIPSI- Dep. Sistemas – Fac. Reg. Santa Fe -Univ. Tecnológica Nacional. Argentina.
Lavaise 610. (3000) Santa Fe
[+]INGAR – CONICET – Fundación ARCIEN
E-mail: hleone@alpha.arcride.edu.ar

En el presente trabajo se desarrolla el metamodelo sobre el cual realizar el manejo de versiones de modelos en una herramienta que soporte análisis y diseño de sistemas de información empleando modelos de objetos. El metamodelo será definido empleando el paradigma de objetos y se utiliza una sintaxis de calculo situacional para expresar formalmente la existencia de un objeto, y su estructura, en una determinada versión del modelo. Uno de los principales objetivos es mantener consistencia y navegabilidad entre los distintos modelos y versiones de los mismos, los cuales se generan durante el proceso de desarrollo de software. La especialización del metamodelo permitirá el manejo de una metodología de desarrollo dado con el grado de desagregación deseado, lo cual se definirá a partir de los elementos definidos en el metamodelo.

El esquema general que se maneja en esta propuesta considera que cada versión del modelo se genera a partir de vistas de un repositorio (Figura) en el que se mantienen todos los objetos que han sido creados y modificados debido a la evolución del modelo. Los elementos que componen el repositorio *R* son las entidades versionables *Ev*, las cuales presentarán una versión *V* (o no, de acuerdo a su existencia en el mismo) en los consiguientes modelos M_i,



$$Versión(Ev, V, M_i)$$

por lo tanto cada modelo será un conjunto de versiones de entidades versionables. El esquema propuesto considera que cada modelo se genera a partir de un modelo precedente. Esta evolución se plantea como una historia compuesta por situaciones discretas, en donde cada nuevo modelo se produce a partir de una versión precedente sobre la cual se aplicó una secuencia de operaciones básicas (*agregar, borrar, modificar, agregar, desagregar*), y la pertenencia de un objeto a una versión será expresado de la siguiente forma.

$$\forall \Phi, Ve, M \; Pertenece(Ve, aplicar(\Phi, M)) \Leftrightarrow ((Pertenece(Ve, M) \vee (agregar(Ve) \in \Phi)) \wedge borrar(Ve) \notin \Phi)$$

Para lo cual se define la función *aplicar*, a partir de una secuencia de operaciones y modelo produciendo un modelo nuevo:

$$aplicar: \Phi \times \mathrm{M} \rightarrow \mathrm{M}$$

Φ*:* es el conjunto de todas las secuencias de operaciones posibles; *M*: el conjunto de los modelos posibles; M_0: modelo inicial. $M_0 = \varnothing$.

Administración de Versiones de Modelos en una Herramienta de Soporte para ADOO

S. Gonnet[*+], R. Holzer[*], H. Melgratti[*] y H. Leone[*+]
[*]GIPSI- Dep. Sistemas – Fac. Reg. Santa Fe -Univ. Tecnológica Nacional. Argentina.
Lavaise 610. (3000) Santa Fe
[+]INGAR – CONICET – Fundación ARCIEN
E-mail: hleone@alpha.arcride.edu.ar

Resumen

En el presente trabajo se desarrolla el metamodelo sobre el cual realizar el manejo de versiones de modelos en una herramienta que soporte análisis y diseño de sistemas de información empleando modelos de objetos. El metamodelo será definido empleando el paradigma de objetos y se utiliza una sintaxis de calculo situacional para expresar formalmente la existencia de un objeto, y su estructura, en una determinada versión del modelo. Uno de los principales objetivos es mantener consistencia y navegabilidad entre los distintos modelos y versiones de los mismos, los cuales se generan durante el proceso de desarrollo de software. La especialización del metamodelo permitirá el manejo de una metodología de desarrollo dado con el grado de desagregación deseado, lo cual se definirá a partir de los elementos definidos en el metamodelo.

1. Introducción

En este trabajo se describen los avances alcanzados en la especificación de las herramientas básicas que permitan mantener los distintos modelos, y sus versiones, que deben ser administrados en un ambiente computacional para ingeniería de software, fundamentalmente empleando un enfoque basado en objetos. Uno de los principales objetivos en un ambiente para ingeniería de software debe ser poder establecer y mantener consistencia entre los múltiples modelos (y sus componentes) que se generan a lo largo del proceso de producción de software (Groth y otros, 1994). La satisfacción de este requerimiento para un ambiente de producción de software debe analizarse en las distintas dimensiones en que se crean los modelos y conceptos a rastrear a lo largo del proceso (Corriveau, 1996).

En primer lugar se puede mencionar que, obviando sus diferencias, las distintas metodologías de análisis y diseño orientado a objetos expresan de diversas formas: la estructura del sistema, a partir de clases y sus relaciones (diagramas de clases, etc.), la interacción entre objetos (diagramas de interacción, de objetos, etc.) y la estructura de comportamiento interno de cada objeto (diagrama de transición de estados, etc.) (Booch, 1994; Jacobson y colab. 1994; Rumbaugh, 1991). Este conjunto de diagramas y modelos, encapsulados en una etapa del proceso de desarrollo de software, representa de alguna manera una de las dimensiones en las cuales es necesario mantener la mencionada navegabilidad y consistencia, la cual en principio sería satisfecha fundamentalmente por la metodología empleada.

Una segunda dimensión a considerar es la originada por las distintas *etapas* o *fases* en que se compone el proceso de desarrollo (análisis, diseño, testing, implementación, etc., las cuales pueden ser subdivididas en más sub-etapas). A lo largo de esta dimensión se producen creaciones, alteraciones, modificaciones, especializaciones o eliminaciones, de los modelos y conceptos empleados en cada fase, para las cuales se requiere mantener las relaciones que permitan rastrear entre fases lo que ha ocurrido. En general el mantenimiento de las relaciones que permiten navegar en esta dimensión no se encuentran representadas en las metodologías.

Finalmente, una tercer dimensión surge al considerar que el proceso de desarrollo de software basado en el paradigma de objetos es inherentemente iterativo (Booch, 1994). En cada iteración se incluye la identificación de requerimientos, análisis, diseño, implementación y testeo. Esta secuencia de etapas, que se repite en cada iteración, puede caracterizarse como un ciclo de vida reducido del sistema. En cada iteración se subsanan errores y se incrementa la comprensión que se tiene del problema. Sin embargo, a medida que se avanza en el proceso iterativo, la secuencia debe converger hacia la solución deseada y por lo tanto cada iteración aumenta su dependencia de la iteración anterior. Aquí surge nuevamente la necesidad de mantener consistencia y navegabilidad entre los modelos y conceptos que se identifican con cada iteración, y la no consideración de las mismas por las metodologías.

En el presente trabajo se desarrolla el formalismo básico que soportará el manejo de las versiones y alteraciones de modelos que se generan en la segunda y tercera dimensión en el proceso de desarrollo de software, en una herramienta de soporte al mismo empleando el modelo de objetos. El metamodelo se especializará de acuerdo a los componentes que considere la metodología en particular que se emplee (por ej.: clases, objetos, atributos, métodos, mensajes, relaciones, etc.), así como las etapas que se consideren para administrar el ciclo de vida del software. En este punto se pretende establecer una estructura sobre la cual luego se impondrán las restricciones referidas a la metodología y ciclo de vida que se deseen emplear.

2. Conceptos Generales

A continuación se desarrolla el esquema propuesto para la administración de versiones de modelos. En éste se considera que cada modelo se genera a partir de un modelo precedente, sobre el cual se han realizado una serie de modificaciones que en general implican la eliminación, creación, modificación, etc., de los componentes del modelo (objetos, clases, relaciones, etc.). A partir de un modelo inicial, pueden construirse sucesivas versiones del mismo. Esta evolución se concibe como una historia compuesta por situaciones discretas, tomándose el modelo del cálculo situacional como marco formal para representar esta forma de modelar el proceso de generación de modelos y versiones. El cálculo situacional fue propuesto originalmente por McCarthy y Hayes en 1962, como un lenguaje de primer orden para razonar acerca de acciones, y si bien ha recibido muchas críticas acerca de su falta de generalidad para representar problemas del mundo real, recientes trabajos que se están realizando a partir del mismo están revirtiendo estas críticas (Pinto y Reiter, 1993). La ontología básica del cálculo consiste en *situaciones*, las cuales corresponden a fotografías del universo estudiado en un instante dado, y *acciones*, que producen evoluciones de una *situación* a otra. Por lo tanto, se considera que cada nueva *situación* es generada a partir de una precedente por la aplicación de una *acción*, en el contexto dado por el proceso de producción de software, cada nuevo modelo que se genera se puede asimilar con una *situación*, y que una *acción* es el conjunto de operaciones que se realizaron sobre el modelo precedente.

3. Esquema Propuesto

Considerando el enfoque presentado en el punto 2., la transformación entre modelos consecutivos puede ser descripta a través de una función que a partir de un determinado modelo, y dada una secuencia de operaciones, genera un nuevo modelo. Llamaremos *aplicar* a dicha función, y es la que en el cálculo situacional, dada una *situación* y una *acción*, denota la *situación* resultante.

$$aplicar: \Phi \times M \rightarrow M \qquad \textbf{(1.)}$$

donde:

Φ: es el conjunto de todas las secuencias de operaciones posibles

M: el conjunto de los modelos posibles.

m_0: modelo inicial ($m_0 = \varnothing$)

Por ejemplo:

$$aplicar(\phi_1, m_1) = m_2;\ \text{donde}\ \phi_1 \in \Phi; m_1, m_2 \in M$$

Una secuencia de operaciones se define de la siguiente manera:

$$\phi = \begin{cases} \lambda & \text{secuencia vacía} \\ o \bullet \phi & \text{donde } o \text{ es una operación} \end{cases} \quad \textbf{(2.)}$$

A partir de la expresión 2. se puede definir por inducción a la función *aplicar*:

$$aplicar(\lambda, m) = m$$
$$aplicar(o \bullet \phi, m) = aplicar(\phi, aplicar(o, m))$$

4. Operaciones Básicas

El término *operación* hace referencia a alguna acción que modifica a un modelo. Se denota genéricamente a cada componente de un modelo con el nombre *Versión de Entidad(v_e).* En este contexto un modelo se define como un conjunto de versiones de entidad. En consecuencia, una operación se define como una transformación de conjuntos. Las operaciones primitivas propuestas inicialmente para representar la transformación de modelos son: *Agregar, Borrar, Simplificar, Refinar y Redefinir.* Las primeras cuatro operaciones permiten incorporar o eliminar conceptos de un modelo, en tanto la última permite crear nuevas versiones de entidades existentes. En principio podrían definirse a las operaciones *Simplificar* y *Refinar* en términos de *Agregar* y *Borrar*, la distinción se realiza, sin embargo, con el objeto de mantener la historia de los cambios realizados en el modelo.

A través de la operación *Agregar* se puede incorporar a un modelo una versión de entidad inexistente en el modelo anterior. Inversamente, la operación *Borrar* elimina una versión existente en el modelo anterior. Una operación frecuente en la generación de modelos es la que permite detallar una entidad a partir de su descomposición en una estructura de nuevas entidades. En este contexto tal operación es denominada *Refinar*, y la misma permite que en el nuevo modelo aparezcan las versiones que desarrollan el refinamiento. La operación inversa a *Refinar* es *Simplificar*, mediante la cual una estructura de entidades (conjunto de *Versiones de Entidad*) se condensan en una versión.

A partir de estas definiciones, se especificará formalmente cuando una *Versión de Entidad* pertenece a un modelo. El predicado *Pertenece(v_e,m)* es verdadero cuando v_e pertenece al modelo *m.* La expresión 3. establece el hecho que el modelo inicial es vacío.

$$\forall v_e \neg Pertence(v_e, m_o) \quad \textbf{(3.)}$$

La expresión 4. establece pertenencia de una *Versión de Entidad*, utilizando el formato de los axiomas de estado sucesor propuesto por Reiter (1991)

$$\forall \phi, v_e, m \quad Pertenece(v_e, aplicar(\phi, m)) \quad \textbf{(4.)}$$
$$\Leftrightarrow$$
$$((Pertenece(v_e, m) \vee (agregar(v_e) \in \phi)) \wedge borrar(v_e) \notin \phi)$$

Refinamiento (v_e, ψ): denota la relación existente entre una versión de entidad v_e de un modelo *m* y un conjunto de versiones de entidades ψ que pertenecen a un modelo

generado a partir de *m*. La expresión 5. expresa formalmente el concepto de *Refinamiento* y su vinculación con la operación *refinar*.

$$\forall \phi, v_e, m_i, \psi, Refinamiento(v_e, \psi) \wedge Pertenece(v_e, m_i) \wedge \psi \subset aplicar(\phi, m_i) \wedge (\psi \cap m_i) = \varnothing \quad \textbf{(5.)}$$
$$\Leftrightarrow$$
$$refinar(v_e, \psi) \in \phi \wedge \neg Pertenece(v_e, aplicar(\phi, m_i))$$

Simplificación(ψ,v_e)*:* denota la relación existente entre un conjunto de versiones de entidad ψ pertenecientes a un modelo *m* y una versión de entidad v_e definida en un modelo sucesor a *m*.

$$\forall \phi, v_e, m_i, \psi, Simplificación(\psi, v_e) \wedge \psi \subset m_i \wedge Pertenece(v_e, aplicar(\phi, m_i)) \wedge \neg Pertenece(v_e, m_i) \quad \textbf{(6.)}$$
$$\Leftrightarrow$$
$$simplificar(\psi, v_e) \in \phi \wedge (\psi \cap aplicar(\phi, m_i)) = \varnothing$$

Redefinición(v_e,,v_e'): denota la relación entre una versión de entidad v_e que pertenece a un modelo m_i y la versión de entidad que resulta de su redefinición v_e' , la cual pertenece a un modelo sucesor de m_i. La expresión 7. expresa la como se vincula la relación *Redefinición* con la operación *redefinir*.

$$\forall \phi, v_e, v_e', m_i \quad \text{Re}definición(v_e, v_e') \wedge Pertenece(v_e, m_i) \wedge \neg Pertenece(v_e', m_i) \wedge Pertenece(v_e', aplicar(\phi, m_i)) \quad \textbf{(7.)}$$
$$\Leftrightarrow$$
$$redefinir(v_e, v_e') \in \phi \wedge \neg Pertenece(v_e, aplicar(\phi, m_i))$$

Las ecuaciones planteadas anteriormente expresan formalmente como determinar el conjunto de versiones de entidad que forman parte de un modelo, lo cual plantea definiciones recursivas a partir de los modelos precedentes. Es decir, para reconstruir un modelo m_{i+1} se deberían aplicar todas las secuencias de operaciones realizadas a partir del modelo inicial para obtener m_{i+1}. Esto surge de:

$m_{i+1} = aplicar(\phi_i, m_i)$; $m_i = aplicar(\phi_{i-1}, m_{i-1})$; ……; $m_1 = aplicar(\phi_0, \varnothing)$
$m_{i+1} = aplicar(\phi_i, aplicar(\phi_{i-1}, aplicar(\ldots aplicar(\phi_0, \varnothing)\ldots))))$
m_{i+1} = aplicar($\phi_0 \ldots \phi_{i-1}\phi_i, \varnothing$); donde $\phi_i\phi_j$ constituye la concatenación de las secuencias ϕ_i y ϕ_j.

Las expresiones presentadas permitirían reconstruir un modelo dado y navegar a través de las operaciones que le dieron origen, y a continuación se presenta de que forma se organizan los objetos que contendrán la información de los modelos generados.

5. Estructura de Objetos que Soporta el Modelo

El esquema general que se maneja en esta propuesta considera que cada versión del modelo se genera a partir de vistas de un repositorio en el que se mantienen todos los objetos que han sido generados y modificados debido a la evolución del modelo. La estrategia surge de la siguiente observación: durante la construcción de un modelo, los distintos conceptos que se incorporan al mismo pueden, en versiones sucesivas: (i) modificarse (e incluso eliminarse) o (ii) asociarse con distintas entidades. Por ejemplo: el cambio de nombre de una clase, una alteración en la signatura de un método, son ejemplos de modificaciones que sufre una entidad. Por otro lado, en distintas versiones de un modelo una clase puede variar sus atributos o métodos, en un diagrama de interacción pueden variar los objetos y mensajes intervinientes.

En base a esta observación se organiza el manejo de versiones de modelos en un esquema de dos capas. Por un lado, un *Repositorio* (*R*) en el que se representan las entidades que surgen durante el proceso de generación de los sucesivos modelos, como así también las relaciones que se establecen entre las mismas. Las entidades que componen al repositorio se denominarán, *Entidades Versionables* (E_v). En la segunda capa, *Versiones* (*V*), están definidas las distintas versiones que se han generado para cada entidad existente en el repositorio, los objetos definidos en este nivel son del tipo *Versión de Entidad* (v_e), empleados en el punto 4. Operaciones Básicas.

Las clases que conforman el repositorio (especializaciones de la clase abstracta *Entidad Versionable* que deberán ser definidas para cada instancia particular del meta modelo) definen los elementos que pueden utilizarse en la construcción de un modelo. Las relaciones que se establecen entre las mismas se corresponden, según la notación utilizada, con las reglas que permiten asociar entidades para formar modelos *sintácticamente* válidos. Es decir, en este nivel se establecen los conceptos a través de los cuales se puede definir un modelo y las asociaciones posibles entre dichos conceptos. Por ejemplo, una instancia de nuestro esquema podría definir, entre otras, a las clases *instancia*, *clase*, *relación*, *método* y *atributo*. Las relaciones a establecer entre estas clases podrían ser: una *clase* ***se compone de*** *atributos* y *métodos*, una *relación* ***asocia*** *a dos o más clases*, etc. Cada instancia del repositorio representa a una entidad introducida en alguna versión del modelo. Las asociaciones que mantiene cada uno de estos objetos permiten determinar el conjunto de entidades con las que se asocia en distintas versiones de un modelo.

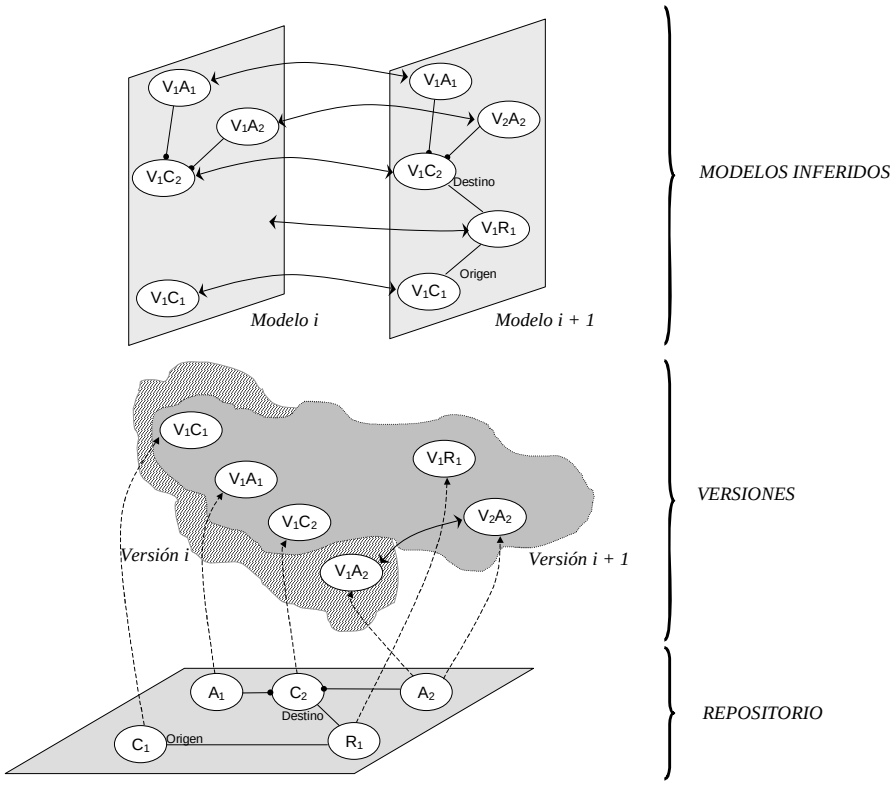


Figura 1

En correspondencia con cada clase del repositorio, se define en el nivel de versiones una clase que especifica las características versionables de dicha entidad. En este sentido, cada versión de entidad define aquellas propiedades de una entidad que pueden ser modificadas en las distintas versiones de modelos. Un conjunto de versiones de entidad constituyen una versión

de modelo. Es así como toda versión de modelo se puede inferir a partir de los niveles de conocimiento antes mencionados, como se esquematiza en la **Figura 1**. De esta manera un concepto formará parte de un modelo si pertenece una versión de dicho concepto al modelo, como lo establece la expresión 4. Por ello algunas entidades del repositorio pueden no formar parte de un modelo (en el caso que ninguna versión de la misma pertenezca al modelo). Además, por razones de consistencia, es necesario restringir a uno la cantidad máxima de versiones de una misma entidad que puedan pertenecer a un modelo.

Luego, la relación que existe entre los dos niveles y un modelo se puede expresar a través de la siguiente definición: cada entidad versionable (instancia de alguna clase componente del repositorio) tiene asociado en un instante del tiempo una o más versiones, pero dado un modelo particular a lo sumo una de dichas versiones pertenece al mismo. Definida la función w: $E_v \rightarrow \wp V_e$; que asocia a cada entidad el conjunto de versiones definidas, es decir: $w(e) = \{v_e / v_e$ es una versión de $e\}$, se debe cumplir la expresión 8.

$$\forall\, e_v \in E_v \mid w(e_v) \mid \geq 1 \qquad \textbf{(8.)}$$

$$\forall\, e_v \in E_v, m_i \in M \mid w(e_v) \cap m_i \mid \leq 1$$

Establecidas las entidades que conforman un modelo se deben determinar las relaciones existentes entre ellas. Primero se debe notar que en la propuesta, las versiones de entidad pertenecientes a un modelo no se asocian **directamente** con otras versiones del mismo modelo, en cambio, como se mencionara anteriormente, las entidades del repositorio mantienen la información de todas las entidades con las que se las ha relacionado durante las distintas versiones de un modelo. En consecuencia, la relación que existe entre dos versiones de entidad se debe inferir a partir de la relación establecida entre las entidades (pertenecientes al *Repositorio*) que las mismas versionan. Se debe notar que, dada la definición del metamodelo, para una versión particular siempre existe una entidad en el repositorio de la cual la misma es versión, es decir:

$$\forall\, v_e \in V_e\ \exists\, e_v \in E_v : v_e \in w(e_v) \qquad \textbf{(9.)}$$

Luego, dada dos versiones de entidad de un modelo, entre las mismas existirá una relación, si y sólo si existe una relación entre las entidades del repositorio que cada una de ellas versionan. Las relaciones que existen entre las versiones de entidades se puede establecer a través del siguiente predicado:

$$\begin{gathered}\forall\, v_{e1}, v_{e2} \in V_{e,}\ m_i \in M, rel: \text{relación}(v_{e1}, v_{e2}, m_i, rel) \\ \Leftrightarrow \\ \text{pertenece}\,(v_{e1}, m_i) \wedge \text{pertenece}(v_{e2}, m_i) \wedge \text{versiona}(v_{e1}, e_{v1}) \\ \wedge\ \text{versiona}(v_{e2}, e_{v2}) \wedge \text{relación}(e_{v1}, e_{v2}, rel)\end{gathered} \qquad \textbf{(10.)}$$

Definida la forma de representar el conjunto de versiones de un modelo, resta explicitar la manera en que se brinda la capacidad de navegabilidad. Dicha capacidad se provee a través de la definición de asociaciones explícitas en el nivel de *Versiones*. Cada operación de transformación que se aplica a una versión de un modelo, incorpora información necesaria para mantener rastros de la evolución de un modelo. Esencialmente, cada operación, además de ejecutar la acción que la misma involucra, establece una relación entre las versiones de entidades a las que se aplica y las que surgen como resultado de su ejecución. El modelo de objetos que representa lo expresado se muestra en la **Figura 2**.

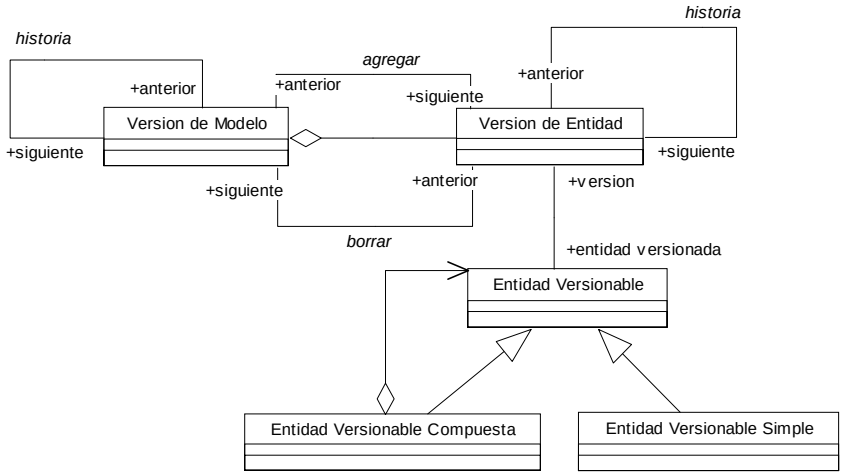


Figura 2: Modelo de objetos

6. Operaciones Básicas en el Modelo de Objetos

Las operaciones definidas inicialmente para el manejo de versiones (*Agregar*, *Borrar*, *Refinar*, *Simplificar*, *Redefinir*) se implementan de la siguiente manera.

Agregar: al aplicar la operación agregar a una versión de modelo, se crea una instancia en el repositorio de alguna subclase de *entidad versionable* y una instancia de la subclase de *versión de entidad* correspondiente. Es responsabilidad de las entidades versionables instanciar las relaciones que mantiene la nueva entidad con las existentes en el repositorio. Se instancia la relación que vincula a una entidad versionada con su nueva versión, y esta última se agrega a la versión de modelo que se está modificando. Por último se deben establecer los vínculos explícitos para mantener la evolución del modelo.

```
agregar: nombre deTipo: tipoEntidad
        |version|
        version := self agregarElemento: nombre deTipo: tipoEntidad.
        version historia: self anterior.

agregarElemento: nombre deTipo: tipoEntidad
        |entidadVersionada version|
        entidadVersionada := tipoEntidad entidadVersionable new: nombre.
        version := tipoEntidad versionEntidad new: nombre.
        entidadVersionada versionadaPor: version.
        self agregarComponente: version.
```

Borrar: La acción de borrar una entidad de un modelo consiste en eliminar la versión de entidad correspondiente del modelo, y en forma similar a *Agregar* establece los vínculos para mantener la evolución del modelo.

```
borrar: versionEntidad
        self borrarElemento: versionEntidad.
        self historia: versionEntidad.

borrarElemento: versionEntidad
        self borrarComponente: versionEntidad.
```

Simplificar: El efecto de efectuar una simplificación consiste eliminar del modelo a las entidades que están siendo simplificadas, y agregar la entidad que se genera como resultado de

la simplificación. Por último se deben establecer los vínculos explícitos para mantener la evolución del modelo.

```
simplificar: listaVersionesEntidad por: nombre deTipo: tipoEntidad
        |nuevaEntidad|
        listaVersionesEntidad do:[x| self borrar: x].
        nuevaEntidad := self agregarElemento: nombre deTipo: nombre.
        listaVersionesEntidad do:[x| nuevaEntidad historia: x].
```

Refinar: Un refinamiento, se puede definir de manera análoga a la simplificación.

```
refinar: version por: listaVersiones
        |nuevasEntidades|
        self borrarElemento: version.
        nuevasEntidades := listaVersiones do: [x| self agregarElemento: x].
        nuevasEntidades do:[x| x historia: version].
```

Redefinir: esta operación reemplaza una versión en el modelo pero genera entidades en el repositorio. Al igual que las restantes operaciones deben mantener los vínculos que provean la capacidad de navegabilidad.

```
redefinir: version por: nombre
        |nuevaEntidad|
        self borrarElemento: version.
        nuevaEntidad := self agregarElemento: nombre deTipo: version tipo.
        version historia: nuevaEntidad.
```

7. Aplicación de la Estructura Propuesta

Con el objeto de generar un modelo capaz de soportar el versionamiento de modelos de Análisis y Diseño OO, se especializa el esquema presentado en la **Figura 2**.

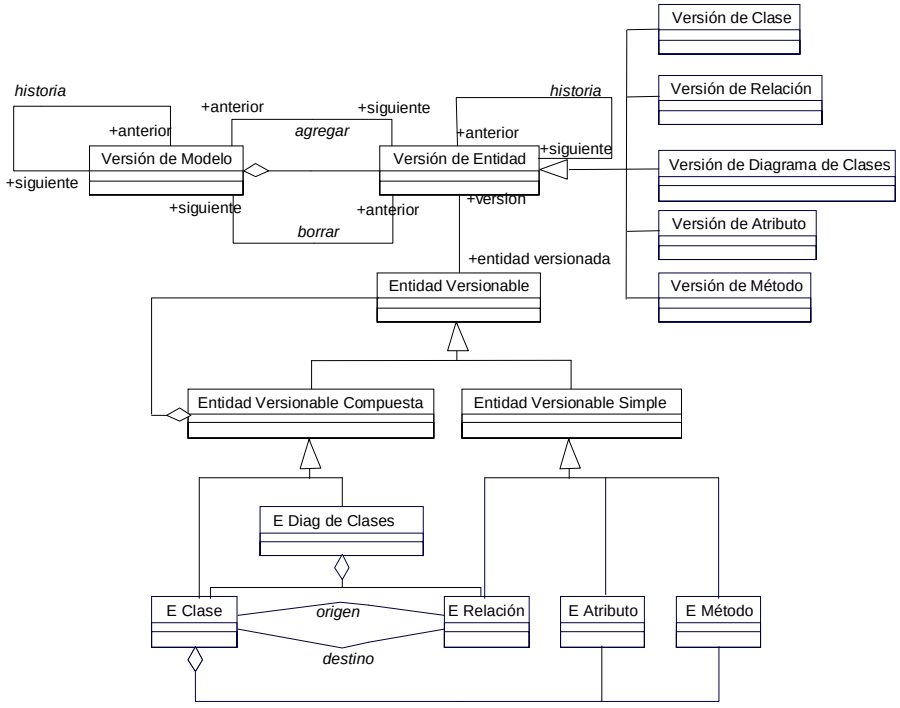


Figura 3

Las entidades de la notación que puedan estar compuestas por otras entidades se definen como una especialización de *Entidad Versionable Compuesta*; el resto de las entidades

especializan la clase *Entidad Versionable Simple*. En correspondencia con cada refinamiento de *Entidad Versionable* se especializa *Versión de Entidad.*

Una especialización (parcial), para el caso particular de una metodología de análisis y diseño orientada a objetos, en la que los modelos están compuestos sólo por Diagramas de Clases, Clases, Relaciones, Atributos y Métodos; el metamodelo resultante se presenta en la **Figura 3**.

Al iniciar la construcción de un nuevo modelo se genera una instancia de *Versión de Modelo* (Versión de Modelo 1) que se corresponde con el modelo vacío inicial. Si se define en el modelo inicial el diagrama de clases presentado en la **Figura 4**, el

Figura 4

Repositorio estará formado por las instancias de las entidades versionables que se muestran en la **Figura 5**. En la misma, se muestran las asociaciones que mantiene cada uno de estos objetos. Las mismas se han establecido a medida que se ha incorporado las entidades versionables.

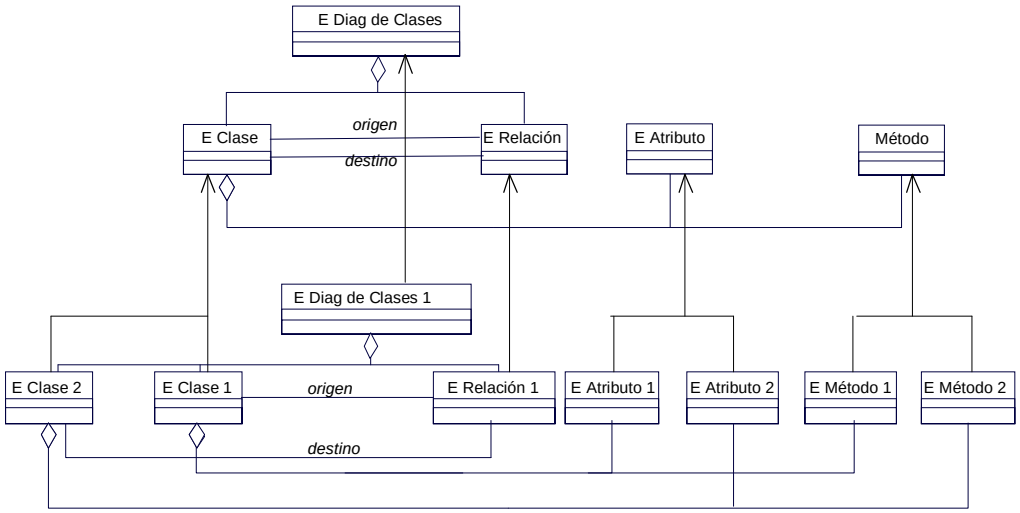


Figura 5

Las instancias generadas en el nivel de *Versiones*, se muestran en la **Figura 6**.

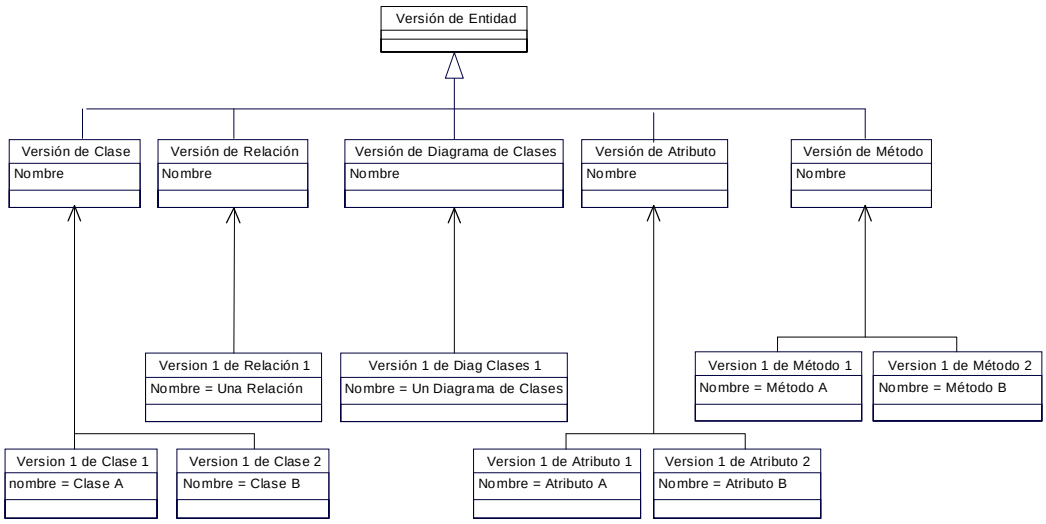


Figura 6

Por último el esquema de instancias generado para representar el modelo planteado en la **Figura 4**, se presenta en la **Figura 7**. Se debe observar que una versión de modelo se compone de versiones de entidades, y no conoce directamente a las entidades del repositorio involucradas en el mismo. Por otra parte, las versiones de entidades no se asocian entre sí a diferencia de las entidades del *REPOSITORIO*. Por último, se hace notar que las entidades versionables se encuentran asociadas respectivamente con sus versiones.

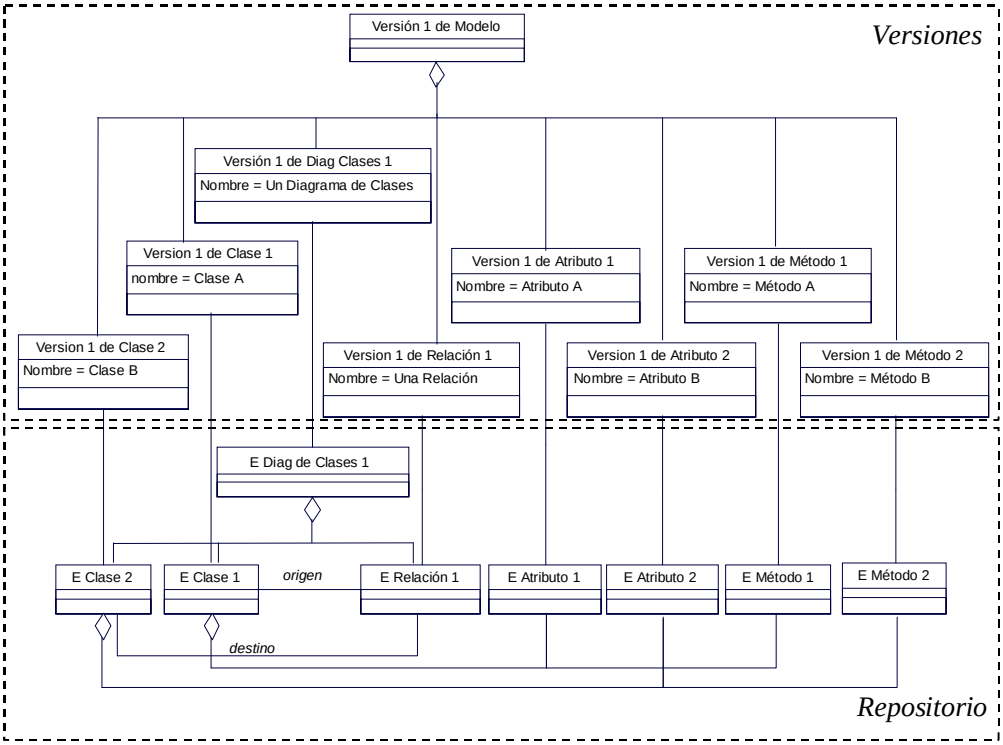


Figura 7

Si se genera una nueva versión del modelo (que se muestra en el diagrama de clases presentado en la **Figura 8**) aplicando las operaciones *Agregar* **Atributo N** a la **Clase A**, *Agregar* **Clase C** y *Agregar* **Otra Relación** con origen en **Clase A** y destino en **Clase C**, al

modelo anterior, se generan en el *Repositorio* las entidades versionables que se muestran en la **Figura 9**.

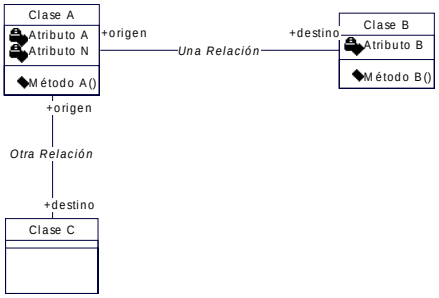


Figura 8

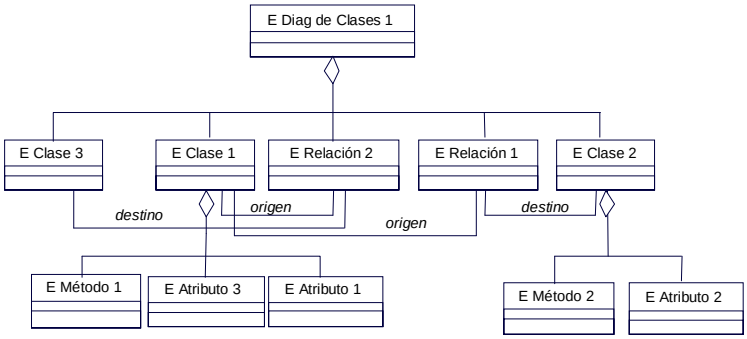


Figura 9

En la **Figura 10** se observan un esquema de las instancias que genera el modelo para representar ambas versiones del modelo. Se debe notar que en el repositorio existen entidades y relaciones que no serán visibles a la Versión 1 de Modelo, dado que no existen versiones asociadas a las mismas que formen parte del modelo (Por ejemplo la entidad del repositorio E Clase 3, que se corresponde a la **Clase C** del modelo). En la misma figura se pueden observar las relaciones que permiten la navegabilidad entre los modelos. Por ejemplo la relación *historia* entre *Versión 1 de Modelo* y *Versión 2 de Modelo*, donde se indica que la Versión 2 es una Versión sucesora a la Versión 1 del Modelo.

8. Conclusiones

Se ha presentado un modelo básico para implementar a partir del mismo el manejo de versiones de modelos en una herramienta que soporte el proceso de desarrollo de software empleando la especificación formal del cálculo situacional y el modelo de objetos. Este desarrollo se encuentra en el marco de un proyecto cuyo objetivo es el desarrollo de un ambiente que soporte integralmente el proceso de desarrollo de software. En el mismo se detectó como una necesidad básica el contar con un esquema que permita manejar consistentemente y navegar a través de los modelos generados en el proceso. Se pretende a través de este modelo documentar la evolución de un modelo de forma tal de poder rastrear cómo entidades identificadas inicialmente se traducen en el modelo final, análogamente, un concepto del modelo final puede rastrearse hacia atrás para entender los criterios de construcción de modelo.

A partir de las operaciones básicas presentadas se pretende modelar las actividades más complejas que se ejecutan en las distintas etapas del proceso de desarrollo de software, de forma tal de registrar su empleo en la generación de las distintas versiones.

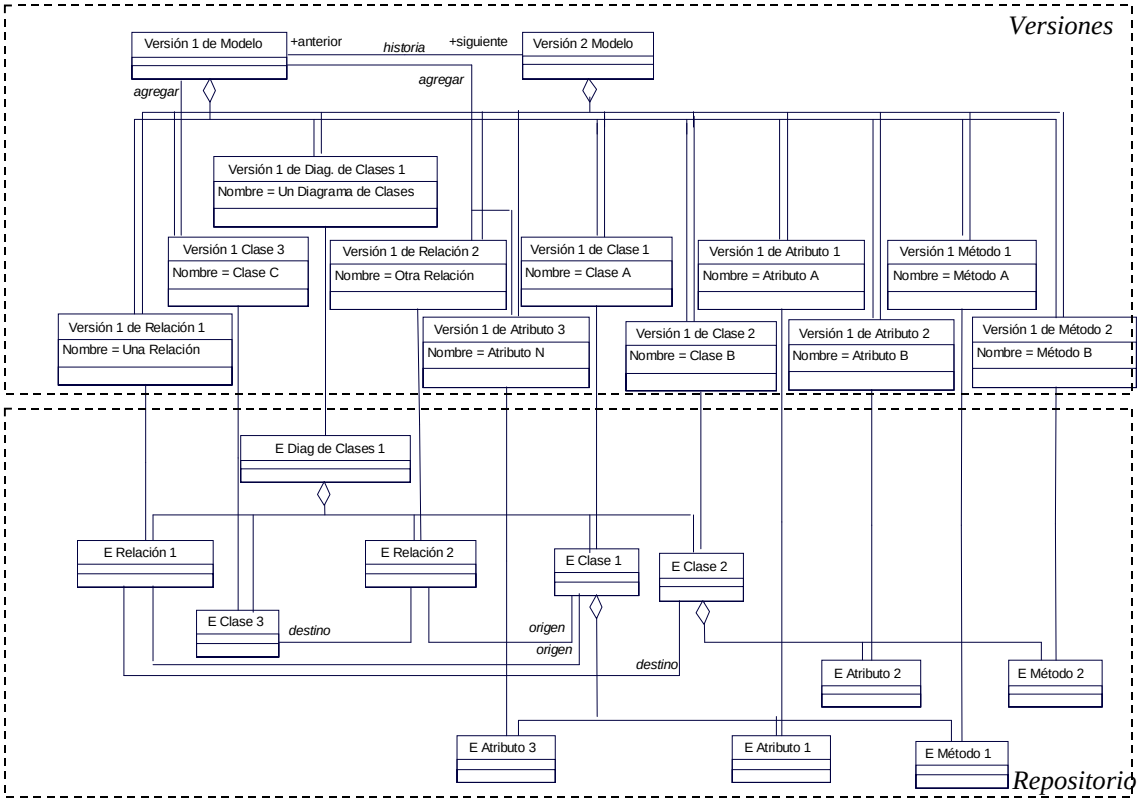


Figura 10

9. Bibliografía

Booch (1994)

Booch G., "Object-oriented analysis and design with applications", Benjamin-cummings, 1994.

Corriveau, 1996

Corriveau, J., “Traceability Process for Large OO Projects”, Computer, **29**, 9, pág. 63 (1996).

Groth y otros, 1994

Groth B., S. Hermann, S. Jähnichen, W. Koch, “Project Integrating Reference Object Library (PIROL): An object-Oriented Multiple View SEE”, Internal Report, Institut für Angewandte Informatik Technische Universität Berlin and GMD-First Berlin (1994).

Jacobson *y colab.* (1994)

Jacobson I., Christerson M., Jonsson P., Övergaad G., "Object-Oriented Software Engineering. A Use Case Driven Approach", Addison-Wesley, 1994.

Martin y Odell (1992)

Martin J. y Odell J., "Principles of Object-Oriented Analysis and Design", Prentice-Hall, 1992.

Pinto y Reiter (1993)

Pinto, J. y R. Reiter, “Temporal reasoning in logic programming: A case for the situation calculus”, Proceedings of the Tenth International Conference on Logic Programming, Budapest, June (1993).

Reiter (1991)

Reiter R., “The Frame Problem in the Situation Calculus: A Simple Solution (sometimes) and a completeness result for goal regression”, Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy, Academic Press (1991).

Rumbaugh.(1991)

Rumbaugh J., Blaha M., Premerlani W., Eddy F., Lorensen W., "Object-Oriented Modeling and Design". Englewood Cliffs, NJ. Prentice Hall, 1991.

Agradecimientos

Este trabajo ha sido financiado en forma conjunta con los proyectos “Ambiente para el Desarrollo de Modelos de Empresas de Producción”, homologado por la Universidad Tecnológica Nacional, y el PIA de CONICET N° 6346. Se agradece el apoyo brindado por estas instituciones.