La Riqueza Formal Presente en el Vínculo entre un Modelo de Bases de Objetos y un Modelo de Transacciones

Clara Smith
LINTI, Universidad Nacional de La Plata, y
Consejo Nacional de Investigaciones Científicas
y Técnicas (CONICET). Buenos Aires, Argentina.
e-mail: csmith@ada.info.unlp.edu.ar

Carlos A. Tau
L.B.S. Informática.
Calle 9 Nro. 1536 ½
(1900) La Plata, Buenos Aires, Argentina.
Fax: +54 21 25-8816

Resumen. Este artículo perfila un vínculo formal preciso entre el modelo para bases de objetos SIGMA y el modelo SAGAS para transacciones no tradicionales, a través de la especificación de axiomas que caracterizan dos regímenes de replicación de cambios: inmediato y tardío. Tanto SIGMA como SAGAS poseen una descripción concisa de sus características, por lo que resulta espontáneo establecer un sólido lazo entre ambos. Sobre la resultante amalgama de estrictas definiciones, es factible construir fértiles y robustas nuevas estructuras, provechosas tanto para la teoría formal de bases de objetos como para la teoría de transacciones.

1 Introducción

Los sistemas de bases de datos tradicionalmente son modelados como colecciones de objetos que sólo pueden ser leídos o escritos por transacciones. El concepto popular de transacción es muy útil, porque oculta a los ojos de usuarios y programadores el procesamiento concurrente y los efectos de las fallas del sistema. El uso de transacciones ha sido muy efectivo en áreas habituales como expendio de pasajes, controles de stock y aplicaciones bancarias. Sin embargo, el amplio uso de las bases de datos en aplicaciones no convencionales (CAD/CAM, automatización de oficinas, ambientes de publicación asistida, información geogáfica y biológica) permitió no sólo el surgimiento de la generación de bases de datos orientadas a objetos, sino también descubrir que el concepto acostumbrado de transacción tiene aplicabilidad limitada. Por ejemplo, en los ambientes mencionados las transacciones necesitan acceder a items de información no usual (como imágenes, audio y video) y los procesos sobre tales datos pueden durar períodos prolongados de tiempo (horas, días y hasta semanas). Estas transacciones, llamadas *larga vida* [Gar87], poseen importantísimas observaciones con relación a las transacciones comunes, o "cortas". Entre ellas, una transacción larga vida es más factible de interrupción debido a su mayor duración. Ante una anomalía, los efectos previos a la misma deben ser deshechos, lo cual es poco aceptable debido a que puede haberse realizado ya mucho trabajo.

En la presente investigación nos concentramos en ciertos aspectos formales presentes en el vínculo proyectado entre un arquetipo de bases de datos orientadas a objetos (o bases de objetos) y un modelo de transacciones no convencionales. En la sección 2 analizamos, el marco de las definiciones formales del modelo SIGMA [ST93], dos regímenes de concreción de evoluciones de esquema -inmediato y tardío- y destacamos que ambos pueden considerarse equivalentes por fallas [Mil89]. Describimos la noción de agente complejo, concepto fuertemente ligado a cierta interpretación específica del régimen tardío: el régimen mix_map. La sección 3 presenta una exhaustiva descripción axiomática de los agentes complejos en el marco de ACTA [Chr90], un soporte axiomático que permite expresar propiedades de transacciones. Los agentes complejos son así formalmente vistos como transacciones atómicas, llamadas x-transacciones. Exponemos en la sección 4 cómo es posible vincular los conceptos de SIGMA con el modelo de transacciones SAGAS [Gar87], no sólo por la naturaleza formal de ambos modelos sino también a través de equivalencias, las nociones de agente complejo, y regímenes de replicación de cambios. Creemos que las definiciones resultantes proveen de una sólida base axiomática y funcional a la teoría de transacciones y regímenes de replicación. A pesar de la elección *ad hoc* de SIGMA, el lector notará que los conceptos presentados de aquí en más se acomodan con naturalidad a cualquier molde típico de bases de objetos. Finalmente, arriesgamos nuestras conclusiones en la sección 5.

2 Esquemas, Bases y Evoluciones

Intuitivamente, un *esquema* de bases de datos reúne las descripciones de los items de información a manejar por un sistema, modelizando entidades físicas y abstractas del mundo real. Los datos concretos que efectivamente existen en un momento dado aparecen en un repositorio físico de información, la *base* propiamente dicha. Toda base se encuentra regida por un único esquema, mientras que un mismo esquema en general puede gobernar varias bases.

Para efectuar modificaciones en esquemas de bases de datos, los usuarios, desde el cuerpo de alguna transacción, invocan alguna *operación de evolución del esquema* presente en el Lenguaje de Manipulación de Datos disponible para el modelo de datos en uso [Ban90][Tau95]. Las bases gobernadas por el esquema modificado son entonces actualizadas mediante la ejecución de primitivas especiales, llamadas comunmente *funciones de conversión* [Kim90] [OOP92] [Fer94], siendo los usuarios ajenos a los mecanismos internos con que se llevan a cabo tales procedimientos. Posteriormente, al activar las bases es posible percibir a través de la

interacción con la información que ciertos cambios han ocurrido: los datos almacenados se vuelven consistentes con las modificaciones efectuadas a alto nivel.

2.1 Decripción Formal de Esquemas y Bases

En el modelo de datos SIGMA un *esquema* es una tupla <*eid*, T, I, A, R>, donde *eid* es el nombre del esquema, T representa el universo de tipos de objetos que conforman dicho esquema, I es el conjunto de implementaciones correctas y concretas de los tipos presentes en T, A es una función que provee axiomas de adecuación o *customización* de los tipos en T (si fuesen necesarios ciertos ajustes), y R es el conjunto de restricciones de integridad impuestas sobre todo el esquema. SCHEMA = (ID x T x I x CUSTOM x AXIOMS) es el dominio asociado a esquemas SIGMA: ID es un dominio primitivo para identificadores; T = [SORTS → SPEC + {*undef*}] es un dominio que a cada nombre de tipo de objetos le asocia un texto de especificación algebraica [Wir86]. El dominio semántico I = [SORTS → PATTERN + {*undef*}] es un dominio de implementaciones que a cada nombre de tipo le asocia un patrón perteneciente al dominio PATTERN = [DATA* → OBJ + {*error*}], donde DATA* es la descripción algebraica del tipo [ST93] y OBJ es un dominio primitivo para identificadores de objetos. Finalmente, los dos últimos dominios de SCHEMA representan dominios primitivos para la representación de axiomas. Una *base* es una tupla <*bid*, π, σ, ζ>, donde *bid* es el nombre de la base, π es un ambiente de composición de objetos (lugar en que se mantienen referencias *parte_de* [Kim89] y de *cliente* entre objetos), σ es el repositorio concreto de datos y ζ es el sistema de clases de objetos que contiene la base. El dominio para bases es BASE = (ID x COMP x STORE x CLASS), donde COMP = [(OBJ x OBJ) → TC + {*unbound*}] es un dominio para ambientes de composición de objetos, en el cual dados dos objetos se indica si uno es parte de otro, STORE representa un dominio para almacenamientos de objetos, y CLASS es un dominio para *clases de objetos*. Por último, descibimos un *sistema de bases de objetos* SIGMA como un grupo de esquemas, donde cada uno de ellos gobierna un conjunto de bases: OBASE_SYS = (SCHEMA x BASE)*)*.

Todos los dominios aquí presentados se encuentran extensamente analizados y detallados en [Tau95].

2.2 Propagación de Cambios

Como un mismo esquema puede servir de referencia conceptual para varias bases, existe la necesidad de diseminar cada una de las evoluciones de esquema sobre las bases concretas que éste gobierna. Tal propagación de cambios debe realizarse sin excepción, de modo de que las bases permanezcan siempre en un estado consistente con respecto a su esquema gobernante. Dicha irradiación puede denotarse por medio de una *operación de* mapeo. Una especificación funcional para la función de mapeo es:

map :: FUNCTION → BASE → BASE**
map fc [ ] = [ ]
map fc b:resto = apply(fc, b):map(fc resto)

Esta definición debe interpretarse como sigue: map es una función de alto orden, que recibe una función (de conversión de datos) y una secuencia de bases, y retorna la secuencia de bases original, donde sobre cada base ha sido aplicada la función de conversión. El símbolo : representa encadenamiento de items. *Apply* es también una función de alto orden [Bir88] que recibe una función y un objeto (una base en este caso), y ejecuta la función usando tal objeto como parámetro real. FUNCTION es un dominio que representa todas las funciones que operan desde un dominio a un codominio [Sch86]; en particular aquí representamos funciones de conversión. El concepto de mapeo de cambios también se encuentra presente en [Mon92].

2.2.1 Mapeo Inmediato y Mapeo Tardío

Siguiendo las dos estrategias más significativas para consumar renovaciones de esquemas [Kim89] [Tre93] [Fer94], podemos dar dos significados distintos al efecto de la función map: *mapeo inmediato* y *mapeo tardío*. El mapeo inmediato consiste en, una vez registrada la evolución en el esquema, replicar instantáneamente as modificaciones sobre las bases subordinadas. En el mapeo tardío los cambios sobre las bases no se realizan en forma abrupta; el sistema los posterga hasta encontrar una ocasión propicia posterior. Por otra parte, un esquema puede sufrir varias renovaciones antes de que sus bases gobernadas sean concretamente actualizadas. Para representar esa situación, asociamos a cada base una *cola de conversión* [Smi95]. Esta cola es una entrada que contiene referencias a funciones, vistas e interpretadas como reformas postergadas, y organizadas según una disciplina FIFO (figura 1). Las colas de conversión son imperceptibles e intangibles por el usuário. Definimos un dominio para describir bases y colas de conversión como (BASE x CQUEUE), ensamblando al dominio BASE un

dominio primitivo CQUEUE para colas. Una implementación del régimen tardío es la política *mix_map* [Smi95] que opera sobre bases y colas de conversión:

mix_map :: FUNCTION → (BASE x CQUEUE) → (BASE x CQUEUE)**
mix_map fc [ ] = [ ]
mix_map fc (b,cc_b):resto = (b, push fc cc_b):mix_map(fc resto)

La política mix-map establece en forma precisa un mapeo inmediato "virtual" de funciones de conversión sobre las bases; *insertándolas* en las colas respectivas (sin acceso súbito a la base). El sistema resolverá con posterioridad cómo aplicar las funciones de conversión encoladas.

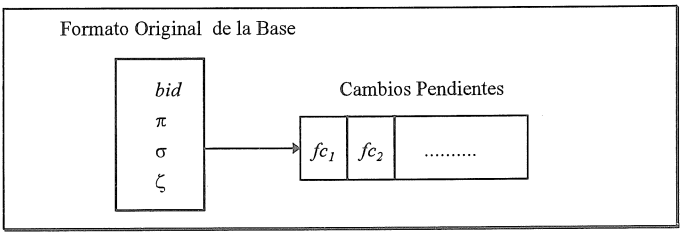

Figura 1: Una Base con su Cola de Conversión

2.2.2 Agentes Complejos y Equivalencia por Fallas entre ambas Políticas de Replicación

Con el nombre de *traza* se conoce a la secuencia de acciones que representa el orden de ejecución de las operaciones presentes en una transacción. Si una transacción falla, es decir, las operaciones en ella no pueden llevarse a cabo satisfactoriamente por determinadas circunstancias (como violación de restricciones de integridad, deadlocks, anomalías de hardware, entre otras), entonces se produce una cancelación de la transacción y las bases no sufren modificación alguna en este caso.

La *equivalencia por fallas* (failure equivalence) de Hoare [Mil89] establece que dos trazas son equivalentes si y sólo si poseen exactamente el mismo conjunto de fallas. Una falla se describe formalmente como un par (R,{1}) donde R es una secuencia de acciones y {1} es un conjunto de nombres de acciones. El par indica que R puede completarse y llegar a un estado en el cual no es posible concretar una acción más si el ambiente de ejecución sólo permite realizar las acciones en {1}. Interesados en las culminaciones exitosas de las transacciones, en un trabajo previo [Smi95] hemos probado que la traza de una transacción ejecutada bajo la política inmediata, es equivalente por fallas [Mil89] a la traza que se obtiene de la misma transacción bajo el régimen mix_map. Para

garantizar la equivalencia por fallas, construimos las trazas asociadas al régimen mix_map utilizando *agentes complejos*. Un agente complejo es un conjunto de acciones organizadas en una unidad de atomicidad, llevando a cabo la totalidad de las operaciones en él o ninguna en caso de falla. De tal modo, las trazas bajo el régimen inmediato y tardío lucen idénticas en estructura, preservando así la equivalencia. El rasgo clave de los agentes complejos es que son visibles al sistema y transparentes para los usuarios.

Definición 2.2.2 *Los agentes complejos son derivados por el sistema a partir de operaciones presentes en transacciones del usuario que operan sobre esquemas y bases SIGMA, y construidos según el siguiente procedimiento:*
- Para la política inmediata, cuando un comando de evolución *ce* sobre un esquema *s* es identificado dentro del cuerpo de una transacción, se construye el agente complejo *<ce s ; map* cf_{ce} *ass_bases(s)>*. El segundo componente simboliza la propagación efectiva e instantánea sobre las bases subordinadas a *s* de la función de conversión asociada al comando *ce*. La función *ass_bases* retorna la lista de bases asociada a *s*. El sistema ejecutará entonces este agente complejo en forma serializada y en una unidad de atomicidad, en lugar de procesar sólo el comando *ce*. Debemos notar aquí que las operaciones de activación de una base, bajo el régimen inmediato, no requieren de la construcción de un agente complejo: la apertura se realiza sobre una única base sin afectar el estado de las restantes.
- Bajo el régimen mix_map, se construyen dos tipos distintos de agentes complejos: i) A partir de una evolución *ce* aplicado sobre un esquema *s*, con la forma *<ce s, mix_map* cf_{ce} *assoc_bases(s)>*. La función *assoc_bases* es una redefinición de la operación *ass_bases*. Su resultado coincide con el dominio semántico definido para la representación del par base-cola. ii) A partir de un comando de activación de una base *ca*, como *<ca b, dequeue b* q_b*>*, donde q_b es la cola asociada a *b* y *dequeue* es:

dequeue :: (BASE x CQUEUE) $\rightarrow$ *(BASE x CQUEUE)*
dequeue b $[\,]_b$ = *(b,* $[\,]_b$*)*
dequeue b frente:resto = ((apply (frente b), [ ]):dequeue (b resto))

Dequeue se ocupa aplicar todas las conversiones sobre la base, de modo súbito. Su definición indica el suministro de la función más antigua y su posterior eliminación de la cola. El proceso se repite con todas las funciones encoladas.

Formalmente, podemos percibir que debido al concepto de equivalencia por fallas, es admisible la coexistencia en un mismo sistema de ambos regímenes de efectivización de evoluciones. Las políticas inmediata y tardía (bajo el régimen mix_map) pueden utilizarse alternativamente para la concreción de los procesos de reforma de bases de objetos. Las colas de conversión resultan un depósito oportuno de secuencias de cambios a realizar sobre las bases, sin interferir la estructura original con que el modelo de datos las concibe. Esto sugiere la facilidad con que cualquier sistema puede adaptarse a la usanza de colas de conversión y adoptar el régimen mix_map.

3 Descripción Formal de los Agentes Complejos

Existen varios marcos formales para establecer el comportamiento dinámico de las bases de datos. *Transaction-Logic* [Qian88], *F-Logic* [Kif89], *ACTA* [Chr90] [Chr94], entre otros. ACTA es un marco axiomático limitado a la lógica de primer orden tradicional. Es por la sencillez y naturalidad en el uso del concepto de transacción que nos

hemos inclinado por utilizar ACTA para formalizar nuestras reflexiones, si bien cualquier otro marco resulta igualmente útil. El modelo de transacciones SAGAS, al que haremos referencia más adelante, también se encuentra axiomatizado por ACTA. Entre sus construcciones básicas, ACTA incluye: i) $p_t[o]$, la invocación de una operación p incluida en la transacción t sobre un objeto o; ii) EO_t, el conjunto de todos los eventos que pueden ser invocados por t ($p_t[o] \in EO_t$), y iii) H, el concepto de historia (o traza) de la ejecución de una transacción o un conjunto de transacciones [Ber87], que contiene todos los eventos asociados, indicando el orden parcial en que ocurren.

3.1 Agentes Complejos como Transacciones Atómicas: X-transacciones

Estamos interesados en describir transacciones atómicas [Lyn94], porque constituyen el bloque básico de construcción de transacciones en la mayoría de los modelos de transacciones. Como deseamos enfatizar también la conveniencia de proveer equivalencia por fallas entre políticas de replicación, cada agente complejo es visto como una transacción atómica, que llamamos x-transacción.

Definición 3.1.1 *Configuración de las x-transacciones. Sea* s *un esquema de bases de objetos SIGMA. Sea* ce *un comando de evolución del esquema definido en el Lenguaje de Manipulación de Datos (LMD) del modelo. Sea* fc_{ce} *una función de conversión asociada a* ce. *Sea* ca *un comando de apertura de base definido en el LMD. Sea* (b, q_b) *de tipo (BASE x CQUEUE) una base con su cola de conversión asociada. Una x-transacción es una transacción atómica que posee alguna de las tres siguientes configuraciones de agentes complejos:*
i) $<ce\ s\ ;\ map\ fc_{ce}\ ass\_bases(s)>$
ii) $<ce\ s,\ mix\_map\ fc_{ce}\ assoc\_bases(s)>$
iii) $<ca\ b,\ dequeue\ b\ q_b>$

Las transacciones atómicas poseen la propiedad de atomicidad por fallas [Chr94]. Fijamos este criterio sobre las x-transacciones:

Definición 3.1.2 *Propiedad de Atomicidad por Fallas de las x-transacciones. Sea* x *una x-transacción,* s *un esquema de bases de objetos SIGMA. Sea* ce *un comando de evolución del esquema,* fc_{ce} *una función de conversión asociada a* ce. *Sea* ca *un comando de apertura de base, el par* (b, q_b) *una base con su cola de conversión asociada. Sea H la historia de un conjunto de transacciones entre las que se encuentra x. Entonces* x *es atómica por fallas, conforme a los axiomas de Atomicidad por Fallas [Chr94], si sucede:*

Ia $\exists s\ \exists ce\ (Commit_x(ce_x[s]) \in H) \Leftrightarrow \exists fc_{ce}\ (Commit_x(map_x[fc_{ce}\ ass\_bases(s)]) \in H))$

IIa $\exists s\ \exists ce\ (Abort_x(ce_x[s] \in H) \Leftrightarrow \exists fc_{ce}\ (Abort_x(map_x[fc_{ce}\ ass\_bases(s)]) \in H))$

Ib $\exists s\ \exists ce\ (Commit_x(ce_x[s]) \in H) \Leftrightarrow \exists fc_{ce}\ (Commit_x(mix\_map_x[fc_{ce}\ assoc\_bases(s)]) \in H))$

IIb $\exists s\ \exists ce\ (Abort_x(ce_x[s] \in H) \Leftrightarrow \exists fc_{ce}\ (Abort_x(mix\_map_x[fc_{ce}\ assoc\_bases(s)]) \in H))$

Ic $\exists b\ \exists ca\ (Commit_x(ca_x[b]) \in H) \Leftrightarrow \exists cq_b\ (Commit_x(dequeue_x[b\ cq_b]) \in H))$

IIc $\exists b\ \exists ca\ (Abort_x(ca_x[b] \in H) \Leftrightarrow \exists cq_b\ (Abort_x(dequeue_x[b\ cq_b]) \in H))$

Esta definición axiomatiza completamente las tres configuraciones características de agentes complejos. Vemos cómo las operaciones de manipulación de la base de objetos (y sólo ellas) constituyen el cimiento para

confeccionar hormas de x- transacciones. Las fórmulas *Ia* y *IIa* hacen alusión a la política inmediata de efectivización de evoluciones: *Ia* establece que la evolución de un esquema ocurre si y sólo si se realiza el mapeo efectivo de la función asociada sobre las bases subordinadas. *IIa* garantiza que la evolución del esquema no puede hacerse efectiva si y sólo si tampoco se realiza el mapeo súbito de las conversiones sobre las bases (ambas acciones abortan). Las expresiones *Ib* y *IIb* se refieren al uso del régimen tardío mix_map: se produce una modificación en el esquema si y sólo si mix_map replica la función de conversión sobre las colas asociadas a cada base subordinada. Asimismo, *IIb* determina que la evolución aborta si y sólo si las acciones de mapeo son abortadas también. Finalmente, la declaración *Ic* expresa que la apertura de una base bajo el régimen mix_map se realiza si y sólo si se procede a la aplicación de las conversiones retenidas en la cola asociada a tal base; *IIc* indica que el fracaso de una apertura conlleva la no actualización de la base a través de la cola de conversión.

3.2 Definición Axiomática de las X-transacciones

La utilización de ACTA para especificar propiedades de los agentes complejos revela la particularidad de éstos de comportarse como un ramillete de conceptos manipulable formalmente. Detallaremos aquellos axiomas que fijan específicamente el comportamiento de las x-transacciones y no de las transacciones atómicas en general. Cabe recordar que la noción de x-transacción se encuentra fuertemente ligada a los conceptos definidos para implantar la política mix_map.

Definición 3.2 *Propiedades básicas de las x-transacciones. Sea x una x-transacción, s un esquema SIGMA; sea el par* (b, q_b) *una base con su cola de conversión asociada. Sea* H_x *la historia de x. El signo* $\rightarrow$ *que aparece en las definiciones siguientes simboliza precedencia.*

1. $ES_x = \{\{ce_i\}, \{ca_j\}, map, mix\_map, dequeue\}$. *Es el conjunto de* eventos significativos *que x puede invocar. Cada* $ce_i \in \{ce_i\}$ *es un comando de evolución del esquema (por ejemplo, en SIGMA las primitivas son: delete, unbound, insert, entre otras [Tau95]). Cada* $ca_j \in \{ca_j\}$ *es un comando de apertura o activación de base del LMD (en SIGMA: set), map y mix_map son las signaturas de las definiciones funcionales asociadas a los regímenes inmediato y tardío respectivamente, y dequeue es la denominación de la función de vaciado de cola (definición 2.2.2).* $ES_x \subset EO_x$.

2. $EI_x = \{\{ce_i\}, \{ca_j\}\}$. *Es el conjunto de* eventos de iniciación *o eventos significativos que pueden invocarse para emprender la ejecución de x.* $EI_x \subset ES_x$.

3. $ET_x = \{map, mix\_map, dequeue\}$. *Es el conjunto de* eventos de terminación *o eventos significativos que pueden ser invocados para la finalización de x.* $ET_x \subset ES_x$

4. x verifica los Axiomas Fundamentales de ACTA.

I. no sucede que x posee en su estructura dos eventos de iniciación.

$\forall e_i \in EI_x ((e_i \in H_x) \Rightarrow \sim \exists e_j \neq e_i \in EI_x (e_j \rightarrow e_i \in H_x)$

II. El hecho de que se concreten mapeos de conversiones, ya sea bajo la política inmediata o la tardía, es debido a que se produjo alguna evolución en el esquema.

$\forall e_i \in EI_x , \forall t_j \in ET_x (t_j \in H_x) \Rightarrow (e_i \rightarrow t_j \in H_x)$

III. no sucede que x culmina por dos eventos de terminación distintos. Una x-transacción no puede ejecutar un mix_map si ha realizado un mapeo inmediato, o viceversa. De igual modo, si en x se produce el vaciamiento de una cola de conversión, no se efectúan mapeos (tardíos o inmediatos).

$\forall\ t_j \in ET_x\ \ (t_j \in H_x) \Rightarrow \sim \exists\ t_k \neq t_i \in ET_x\ (t_j \rightarrow t_k \in H_x)$

5. El conjunto conflictivo de x, contiene todas las operaciones en proceso con respecto a las cuales se debe determinar la presencia de conflictos. Dada x, sean t y t' transacciones cualesquiera, p y p' operaciones del LMD. Analizando por casos:

i) Sea x de la forma <ce s, map fc_{ce} ass_bases(s)>. Las operaciones que conflictúan con x son las que realizan tareas sobre s y las que operan sobre las bases subordinadas a s.

$Conflictivo_x = \{p_t[s] / t \neq x,\ en\_proceso(p_t[s]) \wedge$
$p'_t[ass\_bases(s)] / t' \neq x,\ en\_proceso(p'_t[ass\_bases(s)])\}.$

ii) Sea x de la forma <ce s, mix_map fc_{ce} assoc_bases(s)>, Las operaciones que interfieren con x son las que actúan sobre x y las que proceden sobre las colas de conversión de las bases gobernadas a s.

$Conflictivo_x = \{ p_t[s] / t \neq x,\ en\_proceso(p_t[s]) \wedge$
$p'_t[queues(assoc\_bases(s))] / t' \neq x,\ en\_proceso(p'_t[queues(assoc\_bases(s))])\}.$

iii) Sea x de la forma <ca b, dequeue b q_b >. Los procesos que tienen complicaciones con x son los que trabajan tanto sobre b como los que se ejecutan sobre q_b.

$Conflictivo_x = \{ p_t[b] / t \neq x,\ en\_proceso(p_t[b]) \wedge p'_t[q_b] / t' \neq x,\ en\_proceso(p'_t[q_b])\}.$

En_proceso retorna true *si la acción en cuestión aún no ha finalizado (con o sin éxito).*

6. Si una operación p invocada sobre un esquema, base o cola de conversión finaliza satisfactoriamente, también culmina con éxito la transacción que la invocó; y viceversa:

$\exists o\ \exists p\ (Commit_x\ (p_x[o]) \in H_x) \Rightarrow Commit_x \in H_x$. *El axioma para cada posible forma de x se obtiene trivialmente. La variable o tiene rango SCHEMA + BASE + CQUEUE.*

$(Commit_x \in H_x) \Rightarrow (\forall o\ \forall p\ (p_x[o] \in H_x) \Rightarrow (Commit_x\ (p_x[o]) \in H_x))$ *es la expresión de la forma dual.*

7. $\exists o\ \exists p\ (Abort_x\ (p_x[o]) \in H_x) \Rightarrow Abort_x \in H_x)$; *y*

$(Abort_x \in H_x) \Rightarrow (\forall o\ \forall p\ (p_x[o] \in H_x) \Rightarrow (Abort_x\ (p_x[o]) \in H_x))$ *representan el significado de aquellos incidentes en donde x debe cancelarse ante el fracaso de uno de sus componentes, y viceversa. Las fórmulas para cada configuración de x se consiguen con facilidad.*

8. $\forall o \exists p\ (p_x[o] \in H) \Rightarrow Atómico(o)$. *Todos los objetos sobre los cuales x consuma una operación son objetos atómicos, esto es, detectan sus propios conflictos y poseen mecanismos de locking individuales, como en [Gue95]. Los objetos sobre los cuales x puede proceder son: esquemas, bases y colas de conversión.*

El hecho de interpretar a las x-transacciones como transacciones atómicas permite adoptar el régimen mix_map al accionar de un modelo de transacciones que posea como idea básica de modelización la atomicidad de acciones. En la siguiente sección, concentraremos nuestro esfuerzo en formalizar el enriquecimiento del modelo de transacciones SAGAS con x-transacciones, para operar sobre el modelo de datos SIGMA .

4 SAGAS y X-transacciones: Una Alianza Formal

El modelo SAGAS está basado en el concepto de *acción compensatoria* [Gra81]. Para una transacción *t*, una compensación *c* es una transacción que *deshace semánticamente* los efectos de *t*. Una *saga* es una transacción consistente en un conjunto de operaciones (o subtransacciones) t_1, ..., t_n. Asociada a cada t_i, existen compensaciones c_1, ..., c_n. Para concretar una saga, el sistema debe garantizar la ejecución de la secuencia original t_1, ..., t_k ; o bien la sucesión t_1, ..., t_i, c_i, ..., c_1 para algún $i \leq k$. SAGAS se encuentra caracterizado mediante ACTA, sus axiomas [Chr94] establecen que tanto las transacciones componentes como las compensatorias tienen una semántica similar a las atómicas; especifican el orden de ejecución de las operaciones con respecto a sus compensaciones asociadas y fijan las relaciones entre una saga y las operaciones componentes y compensatorias.

La caracterización formal de x-transacciones puede embeberse en el paisaje de SAGAS, ya que se asimilan como transacciones atómicas. En esta absorción, para el mapeo inmediato producido por el agente complejo $<ce\ s;\ map\ fc_{ce},\ ass\_bases(s)>$ una maniobra compensatoria es $<ce^{-1}\ s;\ map\ fc_{ce}\text{-}1\ ass\_bases(s)>$, donde $ce^{-1}\ s$ indica una operación semánticamente inversa a la evolución *ce* sobre el esquema *s*, de hallarse definida en el LMD (por ejemplo, el par *insert-delete*); y el segundo elemento es la replicación brusca de la operación de conversión $fc_{ce^{-1}}$ ligada a ce^{-1} (provista su existencia). Con el mismo criterio, el agente $<ca\ b,\ dequeue\ b\ q_b>$ tiene eventuales procedimientos reparadores si es viable proporcionar una cola $q^{-1}{}_b = fc_{cen^{-1}} \ldots fc_{ce1^{-1}}$, los contrapesos de $q_b = fc_{ce1} \ldots fc_{cen}$. Esta hechura de $q^{-1}{}_b$ posee la semántica inicial deseada, es decir, la que enmienda todas las acciones a llevar a cabo por las funciones en q_b. Una semántica más laxa puede encontrarse si la alineación de $q^{-1}{}_b$ adolesce de alguna $fc_{cei^{-1}}$. Por último, definimos la compensación para $<ce\ s,\ mix\_map\ fc_{ce},\ assoc\_bases(s)>$ como $<del\_\ mix\ fc_{ce},\ assoc\_bases(s)>$, donde *del_mix* es una función que recibe la lista de duplas base-cola relacionadas a un esquema y elimina la última función ingresada a cada cola:

del_mix :: (BASE x CQUEUE)*→ (BASE x CQUEUE)*
del_mix b [ ]$_b$ = (b, [ ]$_b$)
del_mix (b,q_b):resto = (b, (*delete_last* q_b): *del_mix* resto)

La axiomatización de estas compensaciones se obtiene con proceder análogo al llevado a cabo en la definición 3.2 .

El modelo SAGAS posee diversas extensiones de su versión original (SAGAS de SAGAS, SAGAS con *componentes vitales*, entre otras). La propuesta de enriquecimiento de SAGAS con x-transacciones constituye una nueva disquisición sobre modelo preliminar, que incluso puede acoplarse sobre las variantes existentes. De esta manera, dado un sistema de bases de objetos que se encuentre en condiciones de obrar bajo SAGAS, es factible enriquecer el vínculo entre tal modelo y SAGAS con el concepto de x-transacción, proveyendo un soporte formal para la administración del régimen mix_map. El vínculo formal entre SAGAS y la dupla *mix_map-x-transacciones* se realiza con la unión de los axiomas característicos de SAGAS y los que manifiestan la semántica de x-transacciones. En el presente estudio, el régimen de replicación mix_map utiliza dominios semáticos del modelo SIGMA. La resultante amalgama de definiciones se obtiene sin perjuicio de alterar las semánticas iniciales de ambos modelos ya que se trata de una suma de axiomas que definen casos particulares de transacciones atómicas (las x-transacciones), sin refutar los establecidos para las transacciones atómicas de SAGAS. Así, construimos la

semántica final de nuestra propuesta de manera incremental, combinando especificaciones preexistentes [Wir86]: *Semántica*(Sagas+x-transacciones) = *Semántica*(Sagas) + *Semántica*(x-transacciones) = Definición Axiomática de Sagas ∪ Definición 3.2 ∪ Definiciones para SIGMA [ST93] [Tau95]. Estas últimas involucran dominios semánticos de SIGMA junto con semántica denotacional y operacional de comandos del lenguaje de definición de datos y de manipulación de datos. La elección de SIGMA, como hemos mencionado previamente, es *ad-hoc*. Resulta un modelo de datos óptimo para vincular con SAGAS, debido a su naturaleza formal. Cualquier otro modelo con definiciones estrictas para los dominios de esquemas y bases de objetos pueden adecuarse a la perfección a la usanza de colas de conversión y x-transacciones, sin perturbar la naturaleza de las estructuras originales concebidas por el modelo.

5 Conclusiones

La aplicación correcta de métodos formales que provean una base matemática rigurosa al desarrollo de programas produce resultados de alta integridad. Prueba de ello es la obtención de las definiciones obtenidas para un régimen formal de replicacion de cambios y el concepto asociado de x-transacción, ya que se encuadran en un marco axiomático. Mostramos cómo es posible enriquecer un modelo de transacciones con la noción de x-transacción, de qué modo es factible adaptar el un modelo de datos a la usanza de colas de conversión, y cómo embeber el régimen de replicación mix_map a un modelo de transacciones. Todo ello a través de la constitución, combinación e integración de nociones con fuerte sustento formal. Estamos convencidos de que un régimen de replicación correctamente estipulado puede acoplarse a las especificaciones estrictas provistas tanto por el modelo de datos como por el modelo de transacciones. El enriquecimiento formal incremental sirve de base para la demostración de numerosas propiedades, la definición de equivalencias estrictas y la construcción estratificada de teorías formales que sirvan de sustento para investigaciones y desarrollos futuros.

Referencias

[Ban90] - *Advances in Database Programming Languages*. F. Bancilhon, P. Buneman, eds. ACM Press Frontier Series, 1990.

[Ber87] - *Concurrency Control and Recovery in Database Systems*. P. Bernstein, V Hadzilacos, N Goodman. Addison-Wesley, Reading, MA, 1987.

[Bir88] - *Introduction to Functional Programming*. R. Bird, P. Wadler. Prentice Hall International Series in Computer Science, M. Hoare Series Editor, 1988.

[Chr90] - *ACTA: A Framework for Specifying and Reasoning about Transaction Structure and Behavior*. P. Chrysanthis, K. Ramamritham. ACM SIGMOD International Conference on Management of Data, 194-203, 1990.

[Chr94] - *ACTA: The Saga Continues*. P. Chrysanthis, K. Ramamritham. Database Transaction Models for Advanced Applications, A. Elmagarmid, ed. Morgan Kaufmann Series in Data Management Systems, J. Gray series ed., 350-397.

[Fer94] - *Implementing Lazy Database Updates for an Object Oriented Database System*. F. Ferrandina, T. Meyer, R. Zicari. Proceedings of the 20th International Conference on Very Large Databases, Santiago, Chile. Morgann Kaufmann Publishers, 261-272, September 1994.

[Gar87] - *Sagas*. H. G.Molina, K. Salem. ACM SIGMOD International Conference on Management of Data, 249-259, 1987.

[Gra81] - The Transaction Concept. Virtues and Limitations. J. Gray. Proceedings of the 7th International Conference on Very Large Databases, 144-154, 1981.

[Gue95] - *Modular Atomic Objects*. R. Guerraoui. En TAPOS, *Theory and Practice of Object Systems*, 1 (2), 89-99, John Wiley & Sons, NY, 1995.

[Kif89] - *F-Logic: A Higher-Order Language for Reasoning about Objects, Inheritance and Scheme*. M. Kifer, G. Lausen. ACM SIGMOD International Conference on Management of Data, 134-146, 1989.

[Kim89] - *Features of the ORION OODBS*. W. Kim, N. Ballou. En *Object Oriented Concepts, Databases and Applications*, W. Kim y F. Lochovsky, eds. 251-282, Addison Wesley, 1989.

[Kim90] - *Introduction to Object Oriented Databases*. W. Kim, MIT Press, Cambridge, MA 1990.

[Lyn94] - *Atomic Transactions*. N. Lynch, et al. Morgan Kaufmann Series in Database Systems, CA, 1994.

[Mil89] - *Communication and Concurrency*. R. Milner. M Hoare Series, Prentice Hall, 1989.

[Mon92] - *Lazy Evaluation of Intensional Updates in Constraint Logic Programming*. D. Montesi, R. Torlone. Proceedings of the 2nd International Computer Science Conference ICSC'92, 502-508. Hong Kong.

[OOP92] - *Workshop Report: Objects for Changeable Systems*. OOPSLA'92 Addendum to the Proceedings, 115-121, ACM Press.

[Qian88] - *A Transaction Logic for Database Specification*. X. Qian. ACM SIGMOD International Conference on Management of Data, 243-250, 1987.

[Sch86] - *Denotational Semantics. A Methodology for Language Development*. D. Schmidt. Allyn and Bacon International Editions, MA, 1986.

[Smi95] - *A Formal Lazy Replication Regime for Spreading Conversion Functions over Objectbases*. C. Smith, C. Tau. XXII Seminar on Current Trends in Theory and Practice of Informatics. Milovy, República Checa. Lecture Notes in Computer Science 1012, 455-460, Springer Verlag. Diciembre 1995.

[ST93] - *Un Modelo Unificado para Bases de Datos Orientadas a Objetos*. C. Smith, C. Tau. Trabajo de Grado realizado conforme a los requerimientos establecidos para obtener el título de Licenciado en Informática, Departamento de Informática, Facultad de Ciencias Exactas, Universidad Nacional de La Plata, Argentina. Febrero 1993.

[Tau95] - *Formally Speaking About Schemata, Bases, Classes and Objects*. C. Tau, C. Smith, C. Pons, A. Monteiro. DASFAA'95, 4th International Symposium on Database Systems for Advanced Applications, Singapur. World Scientific Publishing Co., Advanced Database Research and Development Series, Vol. 5, 308-317. Abril 1995.

[Tre93] - *Schema Transformation without Database Reorganisation*. M. Tresch, M. Scholl. SIGMOD RECORD 22 (1), 1993.

[Wir86] - *Structured Algebraic Specifications: A Kernel Language*. M. Wirsing. Theoretical Computer Science, 123-249, 1986.