Gestión de Transacciones en la Plataforma J2EE

Leonardo Rodríguez Viacava
lrodrigu@fing.edu.uy

Daniel Perovich
perovich@fing.edu.uy

INCO – PEDECIBA
Facultad de Ingeniería
Universidad de la República
http://www.fing.edu.uy/inco

Resumen

El manejo transaccional es una característica fundamental de todo sistema de información actual. Los sistemas de información empresariales se caracterizan por integrar fuentes de datos heterogéneas, estar basadas en diferentes tecnologías y tener fuertes requerimientos de performance y escalabilidad. Estas características hacen que lograr transaccionalidad para estos sistemas sea un gran reto. El desarrollo de los mismos se hace muy difícil sin una plataforma que simplifique el trabajo y solucione problemas como el soporte transaccional de los sistemas. La plataforma J2EE es la propuesta de Sun para el desarrollo de aplicaciones empresariales. Esté trabajo apunta a analizar la gestión transaccional que ofrece dicha plataforma.

1 Introducción

El porte de los sistemas de información ha crecido dramáticamente. Sistemas bancarios, sistemas de reserva aérea, entre otros ejemplos, necesitaron extender su alcance, reducir sus costos y bajar los tiempos de respuesta de los servicios. La tecnología utilizada en dichos sistemas no goza, en general, de propiedades deseables del software como mantenibilidad y reusabilidad. A su vez, nuevos requerimientos de performance así como nuevos requerimientos funcionales son los retos que enfrentan estos sistemas. Las empresas han invertido gran capital en dichos sistemas, tanto en la inversión inicial como en el mantenimiento adaptativo y extensional del mismo, lo que lleva a que las empresas consideren en preservarlos.

Las nuevas tecnologías han avanzado en los últimos años. Las metodologías de desarrollo así como las plataformas y frameworks permiten hoy desarrollar software de porte empresarial con tecnologías que antes se utilizaban solamente para sistemas pequeños y medianos. Sin embargo, esta adaptación de las nuevas tecnologías a este tipo de sistemas no es exitosa si no considera el contexto en donde será utilizado. Los nuevos sistemas informáticos de porte empresarial deberán interoperar con los sistemas legados en la empresa, así como interactuar con otros sistemas desarrollados en otras plataformas y mediante otras tecnologías.

La plataforma J2EE es una especificación estándar, que brinda el entorno necesario para el desarrollo de aplicaciones de porte empresarial en múltiples capas. La plataforma soluciona varios problemas esenciales del común de las aplicaciones empresariales (e.g. transaccionalidad, seguridad) simplificando en gran medida el trabajo de los desarrolladores de aplicaciones empresariales. J2EE es una especificación, no un producto de software; esto permite a las empresas optar por los productos de los proveedores de software con los que esta asociados sin necesidad de cambiar de proveedor. Permite a los proveedores de software, además, competir con el resto en función de la calidad de su producto, léase performance, facilidad de uso, cumplimiento del estándar, entre otros. La especificación deja en claro gran parte de los temas relacionados con el desarrollo de aplicaciones de gran porte, dejando otros a consideración de los proveedores de productos.

Uno de los principales retos que enfrentan las empresas al desarrollar nuevo software que interopere con otras aplicaciones existentes es la integración y consistencia de los datos. En muchas empresas se cuenta con fuentes de datos altamente heterogéneas, incluso con redundancia de información en muchos casos. Desafortunadamente otras aplicaciones corren sobre esas fuentes de datos, lo que implica que nuevas aplicaciones no pueden alterar la estructura de la información debido a que dejaría sin funcionamiento a otras aplicaciones legadas. La manipulación de fuentes de datos heterogéneas es un problema que puede ser

subsanado mediante el uso de APIs que encapsulen el acceso a los datos. A pesar de ello, es necesario contar con las garantías que proveen las fuentes de datos en forma independiente a nivel global, esto es, contar con poder transaccional sobre las múltiples fuentes heterogéneas.

La gestión transaccional es un pilar importante en la plataforma J2EE. La misma especifica como llevar adelante transacciones globales sobre múltiples fuentes de datos heterogéneas. Incluso, el poder transaccional no solo radica en las fuentes de datos, sino que es extendido a objetos en memoria, lo cual aumenta fuertemente su poder. El presente trabajo introduce las tecnologías participantes de la plataforma J2EE y discute los aspectos fundamentales respecto de la gestión transaccional en entornos heterogéneos, esto es, consideraciones de autonomía de los sitios y de serializabilidad de las transacciones.

La segunda sección introduce la plataforma J2EE y presenta brevemente las tecnologías participantes, principalmente aquellas readicionadas a la gestión transaccional. La tercera sección discute los aspectos fundamentales respecto de la gestión transaccional; presenta el modelo transaccional de la plataforma, argumenta sobre la autonomía de los sitios, y muestra el tratamiento del control de concurrencia y serializabilidad. Por último, la cuarta sección concluye y presenta trabajos futuros.

2 Especificaciones

La Plataforma J2EE (*Java 2 Enterprise Edition*) define un conjunto de estándares para el desarrollo de aplicaciones empresariales en múltiples capas. La plataforma simplifica estas aplicaciones basándolas en componentes modulares estándar, proveyéndoles un conjunto de servicios a dichos componentes y manejando muchos detalles del comportamiento en forma automática, sin necesidad de programación compleja. Se basa en muchas características de la Plataforma J2SE (*Java 2 Standard Edition*), a saber, portabilidad, acceso a bases de datos (JDBC API), interacción con recursos empresariales existentes (tecnología CORBA), y los modelos de seguridad.

Las aplicaciones empresariales deben ser portables y escalables para ser viables. Generalmente son aplicaciones 24×7, accesibles por miles de clientes simultáneamente. Sin embargo, para este tipo de aplicaciones, desarrolladas en múltiples capas, es difícil establecer su arquitectura. Requieren juntar una variedad de procesos y recursos, datos legados y código legado. En un entorno heterogéneo como el actual, estas aplicaciones deben integrarse a servicios provistos por diferentes vendedores con un conjunto diverso de modelos de aplicaciones y estándares. J2EE rompe la barrera inherente entre sistemas existentes, como ser manejadores de bases de datos, monitores transaccionales, servicios de nombres y servicios de directorios, ubicándose como un estándar por encima de ellos. J2EE engloba los recursos existentes requeridos por aplicaciones empresariales de múltiples capas mediante un modelo de aplicación unificado y basado en componentes [Sun03a].

Un pilar fundamental de toda aplicación empresarial es el soporte transaccional. El servicio de transacciones distribuidas de la Plataforma J2EE involucra cinco participantes diferentes; éstos son descriptos a continuación [Sun99a, Sun99b]. La Figura 1: Servicio transaccional de la Plataforma J2EE presenta estos participantes así como las tecnologías implicadas.

- *Transaction Manager* (TM): encargado de proveer los servicios para demarcación de transacciones, manejo de recursos transaccionales, sincronización y propagación de contexto transaccional.
- *Application Server* (AS): Implementa los servicios necesarios para proveer un entorno de ejecución para las aplicaciones. Entre estos servicios está el manejo del estado transaccional.
- *Resource Manager* (RM): Provee a las aplicaciones acceso a recursos. Este acceso se da a través de *Resource Adapters* (RA)[1]. Este participa de transacciones implementado una interfaz para el manejo de recursos transaccionales, que controla la asociación a una transacción, la finalización de una transacción y la recuperación de una transacción.
- Aplicación basada en componentes transaccionales: Esta aplicación ejecuta sobre un servidor de aplicaciones, el cual le brinda, entre otros servicios, control de transacciones (por ejemplo, una aplicación basada en Enterprice Java Bean).

[1] Un RA es una biblioteca de software que es utilizada por el AS o clientes para acceder a un RM. Típicamente son específicas de cada RM y proveen servicios extras a la conexión como ser la implementación de interfaces para que el RM participe en transacciones distribuidas.

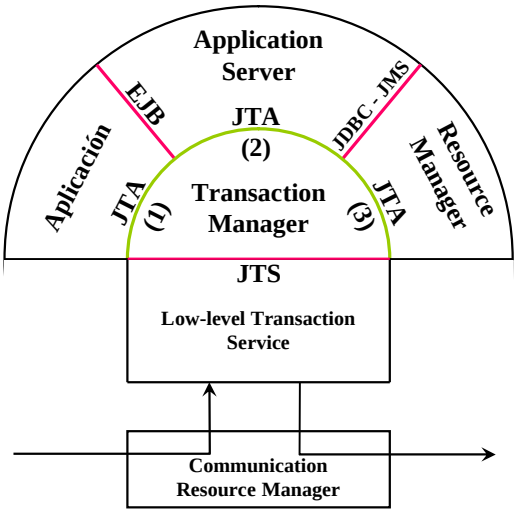

Figura 1: Servicio transaccional de la Plataforma J2EE [Sun99b]

- *Communication Resource Manager*: Encargado de la propagación del contexto transaccional permitiendo acceder a recursos transaccionales ubicados en otros dominios transacciones (estos conceptos provienen de la especificación de X/Open CAE[2]).

La especificación de la plataforma J2EE [Sun01a] a su vez está dividida en diversas sub-especificaciones que permiten especificar los contratos entre los participantes, a saber *Enterprise JavaBeans* (EJB) [Sun02c], *Java Transactional API* (JTA) [Sun99b], *Java Transaction Service* (JTS) [Sun99a], *Java DataBase Connectivity* (JDBC) [Sun01b] y *Java Message Service* (JMS) [Sun02b]. Las siguientes secciones detallarán las mismas.

2.1 Enterprice Java Beans

Enterprise JavaBeans (EJB) es una arquitectura de componentes a nivel de servidor que simplifica el proceso de construcción de aplicaciones de porte empresarial basadas en componentes. EJB no es un producto, es una especificación; como puede verse en la Figura 1: Servicio transaccional de la Plataforma J2EE, es un contrato entre los desarrolladores de componentes y los Product Providers[3]. Los componentes basados en EJB son cargados en application servers (AS), quienes actúan de contenedor[4] para los mismos.

Los contenedores junto con los AS brindan un framework para el funcionamiento de los componentes desarrollados sobre EJB. Dicho framework, cuyas funcionalidades están indicadas en la especificación de EJB, provee de servicios básicos a los componentes, a saber, manejo transaccional, manejo de persistencia, manejo de seguridad, atención simultánea a múltiples usuarios, etc. Teniendo estos servicios ya resueltos, el desarrollador de aplicaciones puede enfocar su esfuerzo en diseñar la solución al problema que esta resolviendo. Así, el diseño obtenido es más puro y estándar dado que hay una clara separación en lo que es la lógica de la aplicación en desarrollo y la lógica necesaria para lograr estos servicios [Sun02c].

Modelo de componentes

En la primera especificación de EJB se propuso dos modelos de componentes, a saber *Entity Bean* y *Session Bean*. La especificación 2.0 introduce un nuevo modelo de componente llamado *Message Driven Bean*. El objetivo de los entity beans es representar entidades con estado persistente. Se los puede ver como la representación orientada a objetos de la información que reside en las fuentes de datos (resources). Por otro lado, el objetivo de los session beans es encapsular los procesos de negocios que conforman al sistema. Un session bean llevará a cabo su función interactuando con otros session beans y con entity

---

[2] X/Open provee un modelo estándar para transacciones distribuidas, potencialmente heterogéneas.

[3] Un Product Provider representa un vendedor de un producto que implementa la especif icación.

[4] El concepto de contenedor es utilizado para los servicios Web y aplicaciones Clientes. Puede encontrarse una descripción detallada en [AAB+02, RAJ02, Sun03a]

beans. Típicamente no son persistentes y pueden participar en transacciones. Por último, los message driven beans tienen como objeto dar soporte a mensajería asincrónica entre componentes [Sun02c].

Manejo transaccional

El soporte del poder transaccional en los sistemas informáticos ha sido bien resuelto a nivel de manejadores de bases de datos [EN00]. En las aplicaciones pequeñas y medianas el manejo de los datos persistentes goza de las propiedades ACID garantidas por los motores de bases de datos que son utilizados en forma directa. A pesar de que este tema está bien resuelto en la actualidad, conlleva a que si el desarrollador necesita mantener información en memoria relacionada con la información en la base de datos, lo que no es usual en aplicaciones pequeñas, esta información deja de estar bajo el control del motor y así se corre los riesgos conocidos de los sistemas que no soportan las propiedades ACID. Algunas técnicas han sido propuestas para resolver este tema; la idea de procedimientos almacenados permite realizar tareas complejas bajo el control del motor de base de datos. Sin embargo esto conlleva a que el desarrollador embeba en la base de datos parte de la lógica de la aplicación, perdiendo portabilidad y reutilización.

Asimismo, cuando una aplicación debe manipular información que radica en distintas bases de datos surge la necesidad de contar con un agente externo que se encargue de monitorear dicha transacción. De esta forma, los monitores transaccionales aparecen como solución a dicho problema [Sun99a].

La plataforma J2EE da un paso en ambas direcciones: poder transaccional a nivel de componentes en memoria y manipulación de información en distintas fuentes de datos. La especificación de EJB incluye consideraciones respecto al manejo transaccional. Existen dos formas de demarcar las transacciones a las cuales el sistema está sujeto, a saber [Sun02c]:

- mediante programación. El propio bean es el que demarca la transacción. Esta forma normalmente se llama *Bean-Managed Transaction* (BMT).
- en forma declarativa. El desarrollador solamente debe establecer en forma declarativa como quiere que el contenedor maneje las transacciones. Esta forma lleva el nombre de *Container-Managed Transactions* (CMT).

Ambos mecanismos tienen soporte para manejo de fuentes de datos heterogéneas, lo cual será explicado en la siguiente sección.

El poder transaccional a nivel de objetos en memoria es logrado por cualquiera de los dos mecanismos. Como dijimos antes, los entity beans son la representación de datos persistentes. El manejo de persistencia puede ser realizado por el propio bean (BMP) o por el contenedor (CMP). La transferencia de datos entre el entity bean y la fuente de datos es llevada a cabo por los métodos `ejbLoad()` y `ejbStore()` [Sun02c]. La sincronización de los datos en memoria con los datos en la fuente de datos es responsabilidad del contenedor. El mismo logra este objetivo invocando `ejbLoad()` en el momento que sea necesario. Así, haciendo `ejbLoad()` cuando una transacción ha abortado se logra el rollback de los datos en memoria.

En cambio, para los session beans el desarrollador debe proveer el código de restauración de la información en memoria. Existen dos tipos de session beans, aquellos que no tienen estado (*stateless*) y aquellos que si lo tienen (*statefull*). En los primeros no es necesaria ninguna restauración al momento de abortar una transacción; para los últimos sí lo es. La especificación permite separar lo que refiere a código de la lógica del negocio de aquel código que permite restaurar el estado en memoria. Si un statefull session bean debe sincronizarse con la transacción en la que participa, debe hacerlo realizando la interfaz `javax.ejb.SessionSynchronization`. La operación `ejbAfterBegin()` de esta interfaz permite hacer una copia de la información al comienzo de la transacción. Las operaciones `ejbBeforeCompletion()` y `ejbAfterCompletion()`, quienes reciben un valor de tipo Boolean como parámetro indicando si la transacción tuvo éxito o no, permiten restauran en caso de ser necesario el estado en memoria utilizando la copia realizada cuando se invocó `ejbAfterBegin()` [Sun02c].

2.2 JTS y JTA

JTA provee un API de alto nivel para el control de transacciones distribuidas. Como se puede observar en la Figura 1: Servicio transaccional de la Plataforma J2EE, JTA hace de mediador entre [Sub99b]:

- El Application Server (AS) y el Transaction Manager (TM) (Número 2 en la figura). Lo usa el AS para controlar las transacciones demarcadas declarativamente y para controlar el enlistamiento de los recursos a las transacciones.
- La aplicación basada en componentes y el TM (Número 1 en la figura). Lo usa el desarrollador de los componentes para demarcar las transacciones manualmente.
- Los Resource Manager (RM) y el TM (Número 3 en la figura). Provee las interfaces basadas en el protocolo XA (estándar de X/Open) a implementar por los resource managers a los efectos de ser enlistados y controlados por el TM.

Un objetivo importante de la especificación J2EE con respecto al manejo transaccional es que sea lo más simple posible. El desarrollador de la aplicación basada en componentes solamente debe demarcar la transacción (en forma manual o declarativa), el resto corre por parte de la infraestructura subyacente. El AS es responsable de enlistar como parte de la transacción a los recursos que los distintos componentes van utilizando y en algunos casos (mencionados en la sección anterior) coordinar para que en caso de rollback el estado de los componentes sea consistente. El TM es encargado de coordinar a todos los RM para realizar el commit o el rollback según sea necesario y cada RM es responsable de hacer lo propio dentro de su dominio. Asimismo (como se explicará más adelante) el AS junto con el TM también son responsables de interoperar (de ser posible) con otros productos J2EE o CORBA-compatibles que estén incluidos en la transacción [Sun01a].

Luego de iniciada una transacción, cada vez que se acceda a un RM, éste será enlistado transparentemente a la transacción. Por ejemplo, si un entity bean CMP que es utilizado por la transacción el AS se encargará de incluir al RM en donde este bean es persistido (disponible a través de los deployment descriptors) en la transacción. Por otro lado, en el caso de los entity beans BMP, cuando éstos piden una referencia al manejador de conexiones de JDBC a los efectos de crear una conexión a un RM, dicho pedido es interceptado por el AS a los efectos de incluir automáticamente dicho RM en la transacción que se está ejecutando. Notar que todo esto siempre sucede en forma transparente para el desarrollador de los componentes.

La especificación nativamente permite que los RM que sean incorporados sean heterogéneos. Para manejar estos recursos soporta JDBC [Sun01b] para bases de datos relacionales y *Java Connector Architecture* (JCA) [Sun02a] para sistemas legados u otro tipo de sistemas. Ambos serán comentados brevemente en la próxima sección.

La especificación de J2EE no exige que los Product Providers soporten algún protocolo particular para interoperabilidad de transacciones entre dos productos J2EE, aunque sean productos idénticos del mismo proveedor. De querer soportar ésto exigen el uso de Internet Inter-Orb Protocol (IIOP) como protocolo de propagación de contexto transaccional. El mismo es un estándar de la Object Management Group (OMG) y está descrito en la especificación de la *Object Transaction Service* (OTS) [OMG01].

JTS especifica la implementación de un TM el cual soporta a un alto nivel las interfaces de JTA e implementa un mapeo con OTS[5] a un bajo nivel. Al implementar OTS, provee propagación de contexto transaccional a través de IIOP que permite interoperabilidad entre JTSs y con otros servidores compatibles con OTS.

2.3 JDBC, JMS y JCA

JDBC es un API parte de J2SE que provee acceso a bases de datos relacionales desde aplicaciones Java. El rol de JDBC dentro del servicio transaccional es permitirle tanto a las aplicaciones basadas en componentes como al AS acceder en forma estándar a RM que sean bases de datos relacionales [Sun01b].

JMS es un API parte de J2EE que provee acceso a servicios de mensajería desde aplicaciones Java. Un servicio de mensajería también se lo ve como un RM, y análogamente a JDBC, JMS permite que las aplicaciones basadas en componentes y el AS accedan a estos RM [Sun02b].

---

[5] Un subconjunto de OTS, para ser más exactos

En las últimas especificaciones de la plataforma J2EE entra en juego también la Java Connector Architecture (JCA), la cual está pensada para hacer posible el acceso a sistemas legados desde AS EJB permitiendo nativamente interoperabilidad a nivel transaccional [Sun02a].

3 Gestión Transaccional

3.1 Modelo Transaccional en J2EE

3.1.1 Modelos Transaccionales

Un modelo transaccional es una forma de realizar las transacciones. Hay diferentes modelos transaccionales, siendo *flat transactions* y *nested transactions* los más populares. Cada modelo agrega su propia complejidad y características a las transacciones del usuario. Para usar un modelo particular, los servicios transaccionales subyacentes deben soportarlo. En la definición de la especificación se acordó que la especificación J2EE solamente exigiría el soporte del modelo de flan transactions dado que el modelo de nested es complejo de implementar y no es ampliamente soportado por todos los manejadores de bases de datos actuales.

Flat Transactions [RAJ02]

Una transacción plana (flan transaction) es una serie de operaciones que son realizadas atómicamente como una unidad de trabajo. Luego que una transacción plana comienza, la aplicación puede llevar a cabo cualquier cantidad de operaciones, ya sea operaciones que repercutan en la persistencia como las que no. Cuando se decide terminar la transacción, el resultado es binario: tiene éxito o falla. Una transacción exitosa es culminada (*committed*), mientras que una transacción que haya fallado es abortada (*aborted*). Cuando se realiza el commit, todos los cambios en la información persistente que hayan realizado las operaciones de la transacción se transforman en cambios permanentes; esto es, todas las actualizaciones realizadas en los recursos de datos, como ser en bases de datos, se hacen 'durables' en almacenamiento secundario solo si la transacción termina con un commit. Si la transacción es abortada, ninguna actualización a los recursos se hacen permanentes ('durables'), y así todos los cambios son revertidos (*roll back*). Cuando una transacción es aborta, todas las operaciones que repercuten en la persistencia que la aplicación llevó a cabo son deshechos en forma automática por el sistema subyacente. La aplicación puede ser notificada en caso de que la transacción aborte de forma de que ésta pueda deshacer los cambios en memoria que ocurrieron durante la transacción. El proceso se describe en la próxima figura.

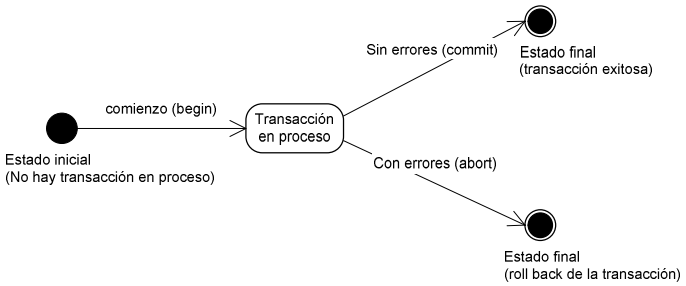

Figura 2: Flat Transaction

Una transacción puede abortar por muchas razones. Muchos componentes pueden estar involucrados en una transacción y cualquiera de ellos puede sufrir un problema que implique que la transacción aborte. Estos problemas pueden ser: parámetros inválidos pasados a uno de los componentes, la violación de un invariante del sistema o falla en el software o en el hardware. Cuando ocurre uno de estos problemas la transacción se marca para abortar. El proceso de abort se realiza en forma automática por el sistema subyacente. Los componentes no incluyen ninguna lógica para deshacer los cambios que fueron realizados durante la transacción. Así, cuando los componentes realizan operaciones dentro de una transacción, deben hacerlo asumiendo que la transacción se completará en forma exitosa.

Otros modelos

Existen otros modelos transaccionales: el modelo *nested transactions* mencionado antes, *chained transactions* y *sagas* [RAJ02]. Este trabajo no profundizará en los mismos dado que la especificación de EJB no brinda soporte para ellos.

3.1.2 Transaccionalidad en Fuentes de Datos Heterogéneas

El concepto de transaccionalidad en fuentes de datos heterogéneas es análogo al de transaccionalidad en bases de datos heterogéneas. Como se mencionó antes, en la plataforma J2EE las fuentes de datos están encapsuladas por un resource adapter. Este nivel de abstracción permite una heterogeneidad de las fuentes de datos aun mayor.

De igual forma que para transacciones en bases de datos heterogéneas, existen dos tipos de transacciones en fuentes de datos heterogéneas: transacciones locales y transacciones globales. Las primeras acceden a datos localizados en una única fuente de datos, mientras que la segunda accede a datos de varias fuentes de datos. La siguiente figura muestra un esquema para transacciones en bases de datos heterogéneas.

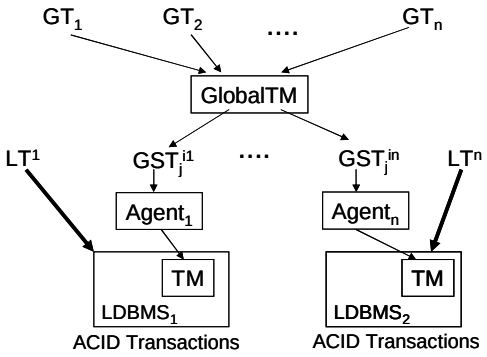

Figura 3: Transacciones en bases de datos heterogéneas [Mot03]

Análogamente, puede construirse un esquema para transacciones en fuentes de datos heterogéneas, basado en las tecnologías involucradas en la plataforma J2EE. El mismo es mostrado en la siguiente figura.

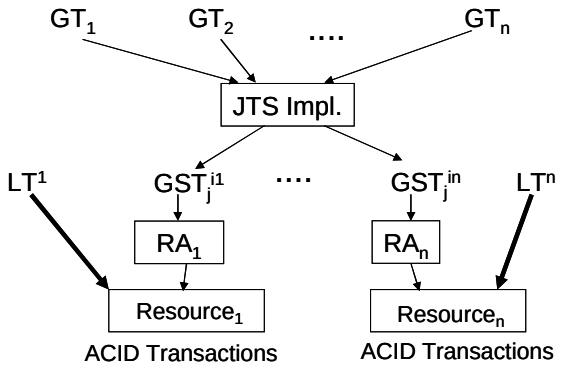

Figura 4: Transacciones en fuentes de datos heterogéneas en la plataforma J2EE

Como se mencionó antes, un Resource Adapter (RA) es una biblioteca de software que encapsula la manipulación de datos manejada por un Resource Manager (RM). Cada RA es enlistado en el estado de la transacción. El Transaction Manager (TM) dirige la transacción. La comunicación entre el TM y el RA es realizada a través de la interfaz `XAResource`. La Figura 5: Arquitectura de Resource Managers presenta la arquitectura del enfoque [Sun99b].

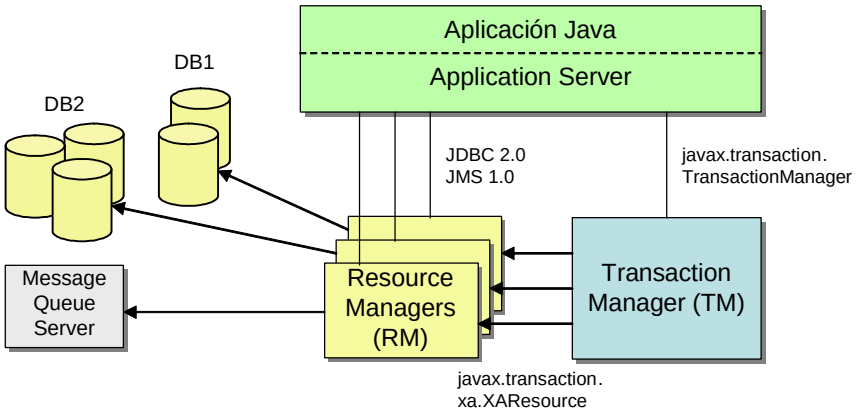

Figura 5: Arquitectura de Resource Managers [Sun99b]

La interfaz `XAResource`[6] es el mapeo en la tecnología Java del estándar de la industria XA basado en especificación X/Open CAE. Esta interfaz define el contrato entre el RM y el TM en un entorno de procesamiento de transacciones distribuidas (DTP). Un RA implementa la interfaz para un RM de forma de permitir asociar una transacción global a una transacción en el resource.

Una transacción global es una unidad de trabajo que se realiza en uno o más RMs en un sistema DTP. Un sistema de estas características se apoya en un manejador de transacciones externo, como ser JTS, para que coordine las transacciones.

La interfaz `XAResource` puede ser soportada por cualquier adaptador de un resource transaccional. Un ejemplo de este tipo de recursos es un RDBMS. Una aplicación puede acceder a los datos a través de múltiples conexiones a la base de datos. Cada conexión es asociada a un RA que hace de proxy con el RM subyacente. El TM obtiene un RA por cada RM que participa en la transacción global. Usa el método `start()` para asociar la transacción global al resource, y el método `end()` para desasociarla. El RM es responsable de asociar la transacción global con el trabajo realizado en los datos entre la invocación a `start()` y a `end()`.

Cuando la transacción esta terminada, los RMs son informados por el TM para hacer prepare, commit o rollback de la transacción de acuerdo con el protocolo de commit en dos fases [Sun99b].

3.2 Autonomía de los Resource Managers

La autonomía de los Resource Managers puede clasificarse en *autonomía de diseño*, *autonomía de ejecución* y *autonomía de comunicación* [Mot03]. Se entiende por *autonomía de diseño* que el manejador de la fuente de datos elija libremente su diseño interno y que la utilización del mismo en transacciones globales no implique realizar cambios en el diseño del manejador. Un manejador presenta *autonomía de ejecución* cuando tiene control total sobre todas las transacciones que ocurren en su sitio, ya sea transacciones locales o globales. Esto incluye la posibilidad de abortar en cualquier momento una transacción. Por *autonomía de comunicación* se entiende que los mecanismos de control de transacciones no son compartidos con el monitor transaccional.

En la plataforma J2EE, los Resource Manager gozan de los tres tipos de autonomía.

En lo referente a la autonomía de diseño, los únicos requerimientos que establece la plataforma sobre los RM es que dispongan de un RA que sirva de mediador entre ambos. Las especificaciones (JTA [Sun99b], JCA [Sun02a]) establecen los requerimientos que deben cumplir los RA para poder formar parte de la plataforma. Estos requerimientos se establecen a través de responsabilidades e interfaces que deben implementar los RA. Esto establece un contrato entre la plataforma (en particular el TM) y los RA. Fuera de los requerimientos que establece el contrato, se dispone de una completa autonomía de diseño tanto para el RA como para el RM.

---

[6] Exactamente `javax.transaction.xa.XAResource`

Respecto a la autonomía de ejecución, los RA son los responsables de enlistar las transacciones globales[7] en los RM cuando el AS así lo requiere. El RA también se encarga de asociar cada solicitud que llegue por la conexión a la transacción global. El RM puede además realizar transacciones locales sobre sus datos. La propia especificación indica que los datos en los resources pueden cambiar por mecanismos ajenos a la plataforma, esto es, desde otras aplicaciones que no estén bajo el control del servidor de aplicaciones o del monitor transaccional. Esto implica que pueda haber varias transacciones ocurriendo en paralelo sobre la misma información. El RM puede optar por abortar cualquier transacción, ya sea local o global. Esto podría darse por ejemplo porque el RM detecta un deadlock o porque al momento del prepare-to-commit el RM vota por rollback. Esto muestra la autonomía de ejecución de los RM.

Por último respecto a la autonomía de comunicación, dado que la única restricción sobre el RM es que cumpla con las interfaces definidas por JTA, las cuales no requieren que el RM deba hacer públicos sus mecanismos de control transaccional, se puede concluir que también se tiene autonomía de comunicación.

3.3 Control de Concurrencia y Serializabilidad

Un aspecto fundamental de una aplicación de porte empresarial es la escalabilidad. Este aspecto plantea un requerimiento importante e ineludible para una plataforma como J2EE, especialmente diseñada para darle soporte a estas aplicaciones. A su vez, la escalabilidad esta fuertemente relacionada al grado de concurrencia que pueda lograr la aplicación para procesar las transacciones solicitadas por sus clientes.

La programación concurrente por si sola trae aparejado un aumento considerable en la complejidad de la programación y en la probabilidad de cometer errores al programar. Una característica interesante de la plataforma J2EE es que el manejo de concurrencia esta a cargo de la infraestructura propiamente. Esto significa que el programador desarrolla sus componentes bajo las reglas de la programación tradicional "single thread" y los detalles complejos son manejados por la plataforma. Esto simplifica considerablemente el trabajo del programador y aumenta la robustez de la aplicación final.

La especificación no define en forma precisa cómo esta concurrencia debe ser resuelta, este es un punto que queda libre para que el Product Provider decida como resolverlo. Típicamente soportan diferentes mecanismos para lograr esta concurrencia, por ejemplo, utilizando pooling de instancias [AAB+02].

Por otro lado también, la concurrencia en combinación con el soporte transaccional introduce varios problemas como: lecturas inconsistentes, lecturas irreproducible y phantom [Mot03]. Estos problemas son bien conocidos, han sido estudiados teóricamente y existen diversas formas de solucionarlos [EN00]. El origen de estos problemas radica en que la ejecución concurrente de dos o más transacciones al final no es equivalente a una ejecución serial de las mismas (serializabilidad) [Mot03]. Una aplicación confiable debe estar exenta de estos problemas.

La especificación J2EE no da una solución completa y estándar para resolver el problema de ejecución de transacciones concurrentes, sino que brinda una serie de lineamientos acerca de cómo debería ser resuelto [Sun01a]. La especificación EJB [Sun02c], por ser la que especifica la capa de negocios (donde generalmente es manipulada la información propiamente) de la plataforma J2EE va a ser la principal afectada por estos problemas. Ésta apunta a solucionar este problema a través de los niveles de aislamiento. Los niveles de aislamiento limitan el grado de concurrencia permitido para las transacciones, a los efectos de eliminar los problemas antes mencionados. Se tienen cuatro niveles de aislamiento, cada uno de ellos elimina uno de los problemas. Cuanto mayor sea el nivel de aislamiento la performance y escalabilidad de la aplicación puede verse más afectada, por lo cual cada aplicación debe definir que nivel de aislamiento requiere a los efectos de optimizar la performance y confiabilidad de la misma.

Los niveles de aislamiento que se pueden manejar son los siguientes [RAJ02]:

- READ_UNCOMMITED. Permite leer información escrita por otra transacción que aún no realizó el commit. No ofrece ninguna garantía de aislamiento, pero ofrece la mejor performance.
- READ_COMMITED. Solamente permite lectura de información de la que se ha realizado commit. Soluciona el problema de lecturas inconsistentes.
- REPEATABLE_READ. Permite solucionar el problema de lecturas inconsistentes y de lectura irreproducible.

[7] En si se la puede considerar como una transacción local asociada a una transacción global.

- SERIALIZABLE. Soluciona los problemas anteriores y el problema phantom. Las transacciones que utilizan este nivel de aislamiento se ejecutan en forma serial. Utilizar este nivel de aislamiento puede afectar considerablemente la performance de la aplicación.

Típicamente los niveles de asilamiento están asociados a bases de datos relaciónales. Los EJB presentan una visión orientada objetos de la información que reside en los RM (normalmente bases de datos relacionales). Los niveles de aislamiento podrían aplicarse tanto a los objetos que residen en memoria como a las bases de datos que los persisten. La especificación EJB "tiende a delegar"[8] toda la responsabilidad del manejo de los niveles de aislamiento a los RM que persisten los objetos en memoria. Según testimonio de uno de los autores de la especificación[RAJ02], el manejar estos problemas a nivel de objetos en memoria es muy complejo de lograr por lo cual se dejó completamente por fuera de la especificación.

La especificación EJB sugiere que los RM sean los responsables de manejar los niveles de aislamiento, pero en ningún momento especifica cómo es que estos niveles de aislamiento deben ser especificados para una aplicación determinada. Sugiere que la responsabilidad de especificar los niveles de aislamiento recaiga sobre:

- Los desarrolladores de los componentes, utilizando APIs propietarias de cada RM o mediante las APIs que provee JDBC. Esto puede ser utilizado por session beans y message driven beans que estén utilizando transacciones manejadas por el bean[9] para establecer el nivel de aislamiento deseado. El API JDBC provee el método `java.sql.Connection.setTransactionIsolation()` para dicho propósito.
- Los deployers, especificarlos declarativamente a través de los deployment descriptors. Esta configuración estaría por fuera de la definición de los descriptors de la especificación. El mecanismo los define el Product Provider. Por ejemplo el servidor de aplicaciones WebLogic de BEA [BEA03] permite definir el nivel de aislamiento que se quiere utilizar para un RM en particular.

Diferentes RMs que participen en una transacción pueden tener diferentes niveles de aislamiento. El intentar cambiar el nivel de aislamiento en medio de una transacción puede ocasionar comportamiento no deseado (cada RM puede reaccionar de forma diferente), por lo cual no debe hacerse. La especificación establece dos restricciones con respecto a esto:

- Los entity beans con CMP los niveles de aislamiento son manejados por las clases generadas por el contenedor. El contenedor debe asegurar que el manejo que hagan estas clases no resulte en conflictos con los niveles de aislamiento existentes para su RM
- Estos conflictos también deben ser evitados cuando se tiene varios entity beans accediendo al mismos RM

Una clara consecuencia directa de esta falta de estandarización de la forma de especificar los niveles de aislamiento es que las soluciones no son portables [RAJ02]. Por otro lado, el hecho de que la responsabilidad de establecer los niveles de aislamiento puede recaer sobre los deployers y que no exista una forma estándar de trasmitir qué niveles de aislamiento son requeridos por los diferentes componentes, lleva a que él mismo deba tener profundos conocimientos sobre los componentes con los que está trabajando a los efectos de no cometer errores que puedan conducir a inconsistencias en la información. Se espera que para futuras versiones de la especificación EJB se llegue a un acuerdo de un mecanismo estándar de especificación de los niveles de aislamiento.

La especificación tampoco define una forma de especificar que tipo de locking se va a utilizar, a saber, optimístico o pesimístico. Diferentes Product Providers pueden soportarlo o no, y definir mecanismos de especificar cual se va a utilizar. Esto nuevamente plantea un problema de portabilidad para la aplicación desarrollada [RAJ02].

---

[8] "Tiende a delegar" porque como se verá más adelante, no define una forma clara y estándar de especificar los niveles de aislamiento.

[9] Los entity beans solo pueden utilizar transacciones manejadas por el contenedor, por lo cual no pueden definir que niveles de aislamiento van a utilizar programáticamente.

3.4 Alcance Transaccional [RAJ02, Sun02c]

Según la especificación EJB [Sun02c], un AS puede soportar transacciones distribuidas involucrando uno o más RM o transacciones locales que involucran un solo RM. Este es un aspecto que depende de cada producto y es transparente al desarrollador. El desarrollador de componentes nunca debe preocuparse si la transacción que se realiza es distribuida o local, el contenedor es el responsable de determinar que tipo de transacción utilizar. Un producto que es lo suficientemente inteligente como para determinar si una transacción va a afectar a un solo RM, puede realizar una optimización y hacer que dicha transacción sea local al RM dejando de lado al TM que causaría un overhead innecesario.

En el caso general de transacciones distribuidas, se pueden tener diferentes escenarios. Un escenario en donde se dispone de un AS el cual interactúa con varios RM o un escenario más complejo en el cual se tienen varios AS los cuales interactúan entre si y a su vez interactúan con varios RM. Este último escenario requiere lograr interoperabilidad entre AS que potencialmente pueden ser de diferentes proveedores. Para ello es necesario disponer de un protocolo de comunicación que les permita coordinar las transacciones, entre otras cosas[10]. Este protocolo se lo llama protocolo de comunicación transaccional (*transactional communications protocol*). La especificación no exige que un AS deba soportar este tipo de interoperabilidad, pero en caso de soportarlo exige que el protocolo a utilizar sea IIOP. Definiendo un protocolo estándar permite que todos los AS que lo soporten puedan comunicarse y participar en transacciones distribuidas. También exige que la coordinación de las transacciones distribuidas sea realizada a través de 2PC (*Two Phase Commit Protocol*).

La información más importante manejada por IIOP es el contexto transaccional. Éste es un objeto que maneja la información de la transacción actual, la identifica y conoce su estado. Éste es compartido por todos los participantes de la transacción y normalmente está asociado al thread de ejecución.

En el caso que se tengan dos AS, en donde uno solicita un servicio al otro de forma que requiera que el contexto transaccional sea propagado y esto no sea posible dado que el AS servidor no soporta propagación de contextos éste debe lanzar una excepción, a los efectos de mantener la consistencia del AS cliente. En el caso inverso en donde se tiene que el AS cliente no soporta la propagación de transacciones y el servidor si, es posible realizar la invocación siempre y cuando el atributo transaccional no exija que el contexto transaccional del cliente deba ser propagado [Sun02c].

4 Conclusiones y Trabajos Futuros

4.1 Conclusiones

En este trabajo se presentó un estudio del soporte transaccional de la plataforma J2EE, analizando el marco general transaccional que presenta las especificaciones de la plataforma, así como los puntos que deja libre para que los diferentes Product Providers puedan diferenciarse. Estas diferencias pueden estar tanto en la calidad de los servicios transaccionales ofrecidos como performance de los mismos.

La infraestructura definida para la plataforma ofrece un amplio conjunto de servicios, a través de los cuales permite acceder a un conjunto importante de fuentes de datos heterogéneas. Las especificaciones que definen como acceder a dichas fuentes heterogéneas (e.g. JDBC, JMS y JCA) establecen interfaces estándares para el acceso a dichas fuentes, así como interoperabilidad a nivel transaccional. Estas fuentes pueden ser incorporadas a la plataforma conservando los niveles de autonomía de los manejadores de las fuentes de datos.

Aparte de la posibilidad de acceso transaccional a fuentes de datos heterogéneas, permite en forma opcional interoperabilidad a nivel transaccional entre diferentes AS provistos (a través de IIOP) por diferentes Product Providers, así como con CORBA. Esto aumenta considerablemente el nivel de distribución, escalabilidad e heterogeneidad de las aplicaciones.

La plataforma soporta mecanismos para el control de concurrencia y serializabilidad de transacciones a través de los niveles de aislamiento principalmente[11]. Estos mecanismos aún no están claramente definidos

---

[10] También puede ser necesario coordinar otros aspectos como la seguridad.

[11] La posibilidad de especificar el tipo de locking a utilizar es una característica muy poco implementada por los proveedores.

por la especificación de la plataforma, lo cual lleva a que quede en manos de los Product Providers definir como especificarlos, afectando directamente a la portabilidad de las aplicaciones empresariales.

4.2 Trabajos Futuros

El presente trabajo analiza el servicio de transacciones de la plataforma J2EE en general. Estudiar estos mecanismos en un producto en particular[12] y analizar como dicho producto cumple los contratos establecidos, permitiría lograr un entendimiento mucho más profundo de la plataforma y del problema general de transacciones distribuidas en fuentes de datos heterogéneas. También permitiría estudiar más en detalle los puntos fuertes y débiles de la plataforma.

En cuanto a la transaccionalidad no se mencionó en ningún momento los mecanismos que ofrece la plataforma para la recuperación ante problemas graves, como la caída de un nodo o un enlace de comunicación. Tampoco se mencionó que problemas se pueden dar cuando se está trabajando en cluster.

La concurrencia y serializabilidad son dos temas que aún requieren maduración en la plataforma. Es interesante analizar que movimientos existen actualmente a los efectos de incorporar una propuesta concreta y completa para resolver estos temas.

También resulta de gran interés estudiar en profundidad JCA como un mecanismo de interoperabilidad transaccional contra sistemas legados u otras plataformas. Por ejemplo como un mecanismo para lograr interoperabilidad con plataformas COM+ y .NET de Microsoft.

5 Bibliografía

[AAB+02] Java Server Programming – J2EE Edition – Editorial: Wrox. S. Allamaraju, K. Avedal, R. Browett, et al.

ISBN: 1-861004-65-6

[BEA03] BEA WebLogic – http://www.bea.com

[EN00] Fundamentals of Database Systems, Third Edition, R. Elmasri, S.Navathe, Addison-Wesley, 2000

ISBN: 0805317554

[JBos03] JBoss Application Server. http://www.jboss.org

[Mot03] Curso de Interoperabilidad – http://www.fing.edu.uy/inco/cursos/interop/interPresencial/

[OMG01] Transaction Service Specification – Versión 1.2 – http://www.omg.org

[RAJ02] Mastering EJB. Ed Roman, Scott Ambler, Tyler Jewell. Editorial: John Wiley & Son.

ISBN: 0-471-41711-4

[Sun99a] Java Transaction Service Specification – Versión 1.0

[Sun99b] Java Transaction API Specification – Versión 1.0.1

[Sun01a] Java 2 Enterprise Edition Specification – Versión 1.3

[Sun01b] Java DataBase Connectivity Specification – Versión 3.0

[Sun02a] Java 2 Enterprise Edition – Java Connector Architecture Specification – Versión 1.5

[Sun02b] Java Message Service Specification – Versión 1.1

[Sun02c] Java 2 Enterprise Edition – Java Enterprise JavaBeans Specification – Versión 2.1

[Sun03a] J2EE Platform Overview - http://java.sun.com/j2ee/overview.html

[12] Un claro candidato es JBoss [JBos03], por ser un AS open source y muy utilizado actualmente.