Especificación y prueba de requisitos de recuperabilidad en transacciones WS-BusinessActivity

Rubén Casado Tejedor[1], Javier Tuya González[1], Muhammad Younas[2]

[1]Departamento de Informática, Universidad de Oviedo, [2]Department of Computing and Electronics, Oxford Brookes University, Oxford (Reino Unido)

<rcasado@lsi.uniovi.es>,
<tuya@lsi.uniovi.es>,
<m.younas@brookes.ac.uk>

Este artículo fue seleccionado para su publicación en **Novática** entre las ponencias presentadas a las VI Jornadas Científico-Técnicas en Servicios Web y SOA (JSWEB2010) celebradas en Valencia y de las que ATI fue entidad colaboradora

Resumen: *Las transacciones son un concepto clave en la fiabilidad de aplicaciones basadas en servicios web, siendo WS-Coordination y WS-BusinessActivity los más recientes y aceptados estándares para su manejo. Este artículo aborda la prueba de transacciones en servicios web, tema al que la investigación actual ha prestado escasa atención. Se define un modelo para la especificación de requisitos funcionales de una transacción según el estándar WS-BusinessActivity, así como se propone la utilización de técnicas basadas en riesgos para la definición de casos de prueba que validen el proceso. Se presenta un caso de estudio para ilustrar el método.*

Palabras clave: *Transacciones, pruebas de software, servicios web, análisis de riesgo*

1. Introducción

Las transacciones son un concepto fundamental para conseguir fiabilidad en aplicaciones distribuidas. Una transacción es un mecanismo que asegura que los participantes en un proceso finalizan de manera consensuada. Tradicionalmente las transacciones siguieron el modelo ACID: Atomicidad, Consistencia, Aislamiento y Durabilidad.

En los Servicios Web (SW), las transacciones son complejas, involucran múltiples participantes, atraviesan diferentes dominios y pueden tener una larga duración. El estricto modelo ACID no es apropiado para un entorno de servicios autónomos e independientes dado que la naturaleza del proceso, (principalmente su incrementada duración) hace inviable el bloqueo de recursos imposibilitando las acciones de *rollback*. Este tipo de transacciones con nuevas características son denominadas de *larga duración o larga ejecución* [14] diferenciándolas de las anteriores, denominadas *transacciones atσmicas*. La comunidad investigadora ha propuesto nuevos modelos de gestión de transacciones que han sido adaptados para los SW. Estos modelos principalmente relajan las *políticas de atomicidad y aislamiento* de manera que los resultados intermedios de las transacciones activas son visibles para otras [24].

Se han propuesto diferentes estándares para su gestión en SW. *Business Transaction Protocol* [5], aun pudiendo ser utilizado en SW, no fue originariamente definido para ello. *Web Services Composite Application Framework* [21] es un estándar para la composición de aplicaciones conteniendo un protocolo para las transacciones. *Web Services Business Process Execution Languaje* (WS-BPEL) [23] es un lenguaje ejecutable para la especificación de interacciones entre SW que incorpora un modelo básico para la gestión de transacciones. *WS-Coordination* (WS-COOR) [22], *WS-AtomicTransactions* (WS-AT) [19] y *WS-BusinessActivity* (WS-BA) [20] son un conjunto de protocolos para la coordinación de transacciones atómicas y de larga duración. Como indica Curbera [11], WS-COOR, WS-BA y WS-AT complementan WS-BPEL proveyéndole de un verdadero mecanismo transaccional para la coordinación de SW.

Aunque la literatura presenta diferentes enfoques para la prueba de SW, existe un vacío en la prueba de transacciones [7]. Nuestra investigación aborda la problemática de la prueba de transacciones de larga duración mediante WS-COOR y WS-BA.

En [8] se presentó un marco conceptual para la prueba de este tipo de transacciones. Dentro de ese trabajo, se estudiaron sus características desde un punto de vista de las pruebas de software. Basado en él, se definió un conjunto de propiedades [9] (*Composición*, *Orden*, *Visibilidad*, *Consistencia*, *Durabilidad*, *Coordinación*) donde aplicar análisis de riesgos con el objetivo de identificar escenarios de prueba. Se muestra el uso de la técnica *Fault Tree Analysis* (FTA) [18] para definir especificaciones de casos de prueba.

En este artículo extendemos el trabajo anterior mejorando el modelo para la especificación de los requisitos transaccionales así como la adaptación del enfoque basado en riesgos para WS-BA. La utilidad se ilustra con un caso de uso donde los requisitos funcionales se especifican guiados por nuestro modelo y se derivan dos casos de prueba usando los escenarios de riesgos alcanzados en el FTA.

El resto del artículo se organiza así: La **sección 2** resume el funcionamiento de WS-COOR y WS-BA. La **sección 3** presenta el modelo y notación para la especificación de los requisitos transaccionales. En la **sección 4** se ilustra nuestro enfoque con un caso de estudio. Este caso de estudio se usa para mostrar cómo los casos de prueba para WS-BA se derivan de los escenarios de riesgo en la **sección 5**. La **sección 6** compara nuestro trabajo con otras investigaciones. Algunas conclusiones y líneas de trabajo futuro se describen en la **sección 7**.

2. WS-Coordination y WS-Business Activity

WS-COO define un protocolo para la distribución del contexto de una transacción a todos los participantes. WS-COO especifica la interfaz del coordinador creando uno nuevo o uniéndose a una transacción ya existente. El coordinador consta de los siguientes servicios:

- Activación: creación de un nuevo contexto.
- Registro: especificación del protocolo a usar.
- Protocolo: tipo de coordinación (WS-BA para transacciones de larga duración y WS-AT para las atómicas).

Cuando un participante inicia una nueva transacción necesita un nuevo contexto que solicita a un coordinador usando el servicio de activación. Posteriormente, el participante notifica al resto reenviando el contexto de coordinación. Cada participante registra su protocolo en un coordinador usando el servicio de registro. Finalmente, el coordinador gestiona la transacción intercambiando mensajes con los participantes mediante el servicio de protocolo.

Tanto WS-AT como WS-BA están construidos sobre WS-COOR. El propósito de WS-BA es coordinar transacciones de larga duración, compuestas por subtransacciones, utilizando compensaciones. Una vez que una subtransacción finaliza correctamente puede ser deshecha mediante la ejecución de una compensación. Imaginemos una aplicación dedicada a la reserva de un viaje compuesto por reservas de hotel, vuelo y alquiler de coche, donde cada reserva la realiza un servicio independiente. Si el vuelo se anula una vez que el hotel fue reservado, esa reserva necesita ser cancelada. Esto significa que la habitación tiene que aparecer como disponible y el cliente podría tener que pagar una penalización por la cancelación.

3. Modelo de transacción

Una transacción de larga duración en SW, denotada por *wT*, está estructurada como *saga* [13], un conjunto de subtransacciones cada cual con una acción compensatoria asociada. Si una de las subtransacciones de la secuencia aborta, se ejecutan, en orden inverso, las acciones compensatorias asociadas a todas las subtransacciones completadas. Se ve la importancia de comprobar la *recuperabilidad* de cada subtransacción para asegurar la consistencia del proceso. En esta sección se propone un modelo y notación para la especificación de los requisitos funcionales transaccionales. Este modelo será usado para la definición de casos de prueba.

Una *wT* se compone de actividades (subtransacciones) ejecutadas por diferentes SW (participantes) que pueden necesitar una cantidad substancial de tiempo para terminar su acción. Una *wT* queda definida por la quíntupla $wT = <S, C, \Psi_I, \Psi_E, \Psi_C>$ donde S es el conjunto de subtransacciones, C el de acciones compensatorias, Ψ_I es un conjunto de *estados iniciales*, Ψ_E es un conjunto de *estados ejecutados* y Ψ_C es un conjunto de *estados compensados*.

El conjunto $S=\{s_1, ..., s_n\}$ define las subtransacciones donde cada s_i es una transacción atómica u otra *wT'*. Una subtransacción es reemplazable si existe otro participante que pueda realizar la misma acción. Una subtransacción s_j es dependiente de otra s_i, denotado por $s_j < s_i$ si puede ser ejecutada si y solo si s_i fue completada.

S tiene un orden de ejecución, denotado por $O(S)$. Se denota $s_i ; s_j$ si la subtransacción s_i tiene que finalizar antes de poder ejecutar s_j. Se denota s_i / s_j si ambas subtransacciones pueden ser ejecutadas en paralelo. Se usa $[s_i / s_j]$ para especificar subtransacciones reemplazables que pueden ser ejecutadas en paralelo y finalmente escoger cuál de ellas confirmar. Se usa $[s_i : s_j]$ para especificar que s_i y s_j son reemplazables y s_j puede ser ejecutada si y solo si s_i falló. Si estrictamente solo una puede ser ejecutada, se denota mediante $[s_i , s_j]$. La **figura 1** muestra la notación gráfica para estas relaciones.

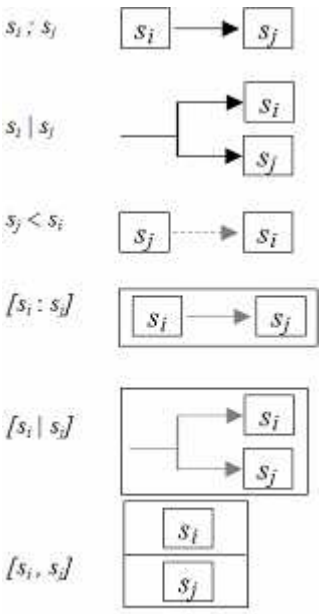

Figura 1. Relaciones entre subtransacciones.

Toda s_i tiene una *acción compensatoria* denotada por c_i. Una c_i deshace, semánticamente, las acciones realizadas por s_i, pero no necesariamente retorna al sistema al estado previo a su ejecución. El conjunto de todas las acciones compensatorias se denota por $C=\{c_1, ..., c_n\}$. Toda c_i puede ser ejecutada si y solo si s_i ha sido completada. Cualquier c_i puede ser una acción vacía, denotado por λ.

3.1. Participantes y coordinador

Un participante p_i es el servicio responsable de ejecutar s_i y la gestión de c_i.

Una *notificación de transacción* es la comunicación entre dos participantes. Se usa la notación $i_{wT}[m_1]j_{wT}$ para especificar que el participante p_i interviene en *wT* y envía el mensaje m_1 a otro participante p_j también involucrado en *wT*. Si el participante es el coordinador se denota mediante *K*. Se usa $i_{wT}[m_1]\, j_{wT} - l_{wT}[m_2]O_{wT} - ... - v_{wT}[m_n]z_{wT}$ para denotar una secuencia de notificaciones.

Un *coordinador K* es el participante encargado de manejar las subtransacciones. Ordena su ejecución, gestiona los fallos y compensaciones y recolecta los resultados de los participantes para dejar el sistema en un estado consistente después de la ejecución de la transacción.

3.2. Estados en una subtransacción

Un *estado inicial* σ_i define los requisitos necesarios para que p_i pueda ejecutar s_i. El conjunto de estados iniciales se denota $\Psi_I = \{\sigma_1, ..., \sigma_n\}$.

Un *estado ejecutado* σ^*_i define los requisitos que p_i tiene que satisfacer una vez que ha completado s_i. El conjunto de todos los estados ejecutados se denota por $\Psi_E = \{\sigma^*_1, ..., \sigma^*_n\}$.

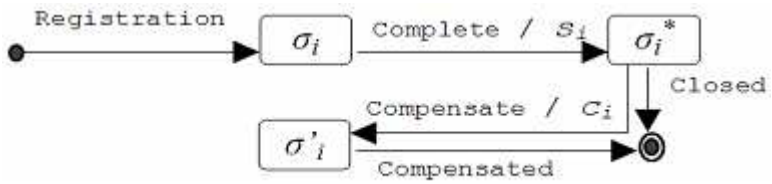

Figura 2. Estados de una subtransacción.

Los requisitos especificados en σ^*_i pueden estar incluidos en los requisitos necesarios para la ejecución de la siguiente subtransacción s_{i+1} definidos en σ_{i+1}.

El estado compensado σ'_i define los requisitos que p_i tiene que satisfacer una vez ha ejecutado c_i. El conjunto de todos los estados compensados se denota por $\Psi_C = \{\sigma'_1, ..., \sigma'_n\}$.

La *acción compensatoria* c_i deshace semánticamente las acciones ejecutadas por s_i. El sistema cambia hacia σ'_i. Este estado no es necesariamente igual a σ_i dado la imposibilidad de deshacer algunas operaciones (por ej. un *email* enviado). Se usa

$$\sigma_i \xrightarrow{s_i} \sigma_i^*$$

para denotar que, empezando en σ_i, se alcanza σ^*_i después de que la subtransacción se completa y la secuencia de ejecución puede continuar. Se usa

$$\sigma_i^* \xrightarrow{s_i} \sigma_i'$$

para denotar que, empezando en σ^*_i, se alcanza σ'_i tras la ejecución de c_i y se puede continuar con la secuencia de compensaciones.

La **figura 2** ilustra los estados de una subtransacción. Se puede hacer un fácil mapeo entre el modelo de requisitos con el modelo de estados de WS-BA [20]. Nótese que σ_i puede ser mapeado como el estado *Active* ya que es el estado anterior a la ejecución de la subtransacción (mensaje *complete*). De la misma manera, σ^* está relacionado con el estado *Completed* mientras que σ'_i es alcanzado a través del mensaje *compensated*.

4. Caso de estudio

Servicio de Estancias (*SerEs*) es una aplicación basada en SW dedicada a la gestión de las solicitudes de estudiantes para hacer estancias en otras universidades. El servicio proporciona completa funcionalidad para la asignación de las becas, registro y reserva de alojamiento en la universidad destino así como del pago.

El estudiante envía a la aplicación sus datos personales, fechas para la estancia y una lista ordenada por preferencia de las universidades en las que está interesado. La aplicación envía simultáneamente la solicitud al servicio de

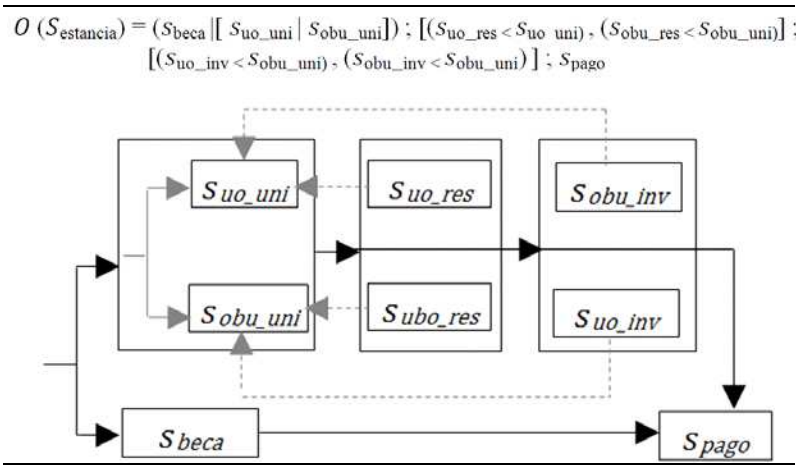

Figura 3. Relación y orden de ejecución de las subtransacciones.

becas así como a todas las universidades (*uni*) propuestas. Después de ser analizada, el servicio de de becas responde con los detalles de la subvención que variará según la duración y lugar de la estancia. Obviamente la subvención puede ser denegada. Cuando todas las universidades han respondido, la aplicación selecciona la mejor acorde con las preferencias del estudiante y notifica la decisión al resto de universidades candidatas. Si la subvención es denegada, o todas las universidades candidatas rechazan la solicitud, el proceso queda cancelado y se notifica al estudiante. Una vez que la universidad acepta la propuesta y se asigna la cuantía de la beca, el estudiante es matriculado en la universidad destino. Esta acción se compone de reserva de alojamiento en una residencia universitaria (*res*) y alta en el servicio de investigación (*inv*). El último paso es cargar el pago de la residencia universitaria en la cuenta del estudiante y notificar al usuario todos los datos de la estancia.

En este ejemplo existen múltiples dependencias *inter* e *intra* transaccionales. De ahí la importancia de asegurar la consistencia tanto durante como al final del proceso, tanto si finaliza correctamente o es cancelado (y recuperado mediante acciones compensatorias). Incluso asumiendo que todos los servicios funcionan correctamente, existen escenarios de riesgo debidos a la naturaleza transaccional del proceso. A continuación se presentan algunos:

- Si un estudiante con beca la rechaza y existen otros estudiantes esperando por una, ¿se concede una nueva beca?
- Si varias universidades candidatas aceptan la propuesta, ¿se elige la preferida? ¿Cancelan el resto la propuesta?
- Si después de que todo el proceso esté confirmado el estudiante acorta la duración de la estancia, ¿se reclama la parte correspondiente de la subvención? ¿Cancela la residencia de estudiantes el alojamiento?

4.1. Especificación de requisitos

En este apartado se especifican los requisitos funcionales transaccionales para *SerEs*. Se asume que el estudiante estaría interesado en hacer la estancia en la Universidad de Oviedo (UO) o en Oxford Brookes University (OBU). Esta transacción se define a continuación:

- $wT_{estancia} = S_{estancia}, C_{estancia}, \Psi_{Iestancia}, \Psi_{Eestancia}, \Psi_{Cestancia}$
- $Participantes = \{Beca, UO_{uni}, OBU_{uni}, UO_{res}, OBU_{res}, UO_{inv}, OBU_{inv}, Pago, Coordinador\}$
- $S_{estancia} = \{s_{beca}, s_{uo\_uni}, s_{obu\_uni}, s_{uo\_res}, s_{obu\_res}, s_{uo\_inv}, s_{obu\_inv}, s_{pago}\}$
- $C_{estancia} = \{c_{beca}, c_{uo\_uni}, c_{obu\_uni}, c_{uo\_res}, c_{obu\_res}, c_{uo\_inv}, c_{obu\_inv}, c_{pago}\}$
- $\Psi_{Iestancia} = \{\sigma_{beca}, \sigma_{uo\_uni}, \sigma_{obu\_uni}, \sigma_{uo\_res}, \sigma_{obu\_res}, \sigma_{uo\_inv}, \sigma_{obu\_inv}, \sigma_{pago}\}$
- $\Psi_{Eestancia} = \{\sigma^*_{beca}, \sigma^*_{uo\_uni}, \sigma^*_{obu\_uni}, \sigma^*_{uo\_res}, \sigma^*_{obu\_res}, \sigma^*_{uo\_inv}, \sigma^*_{obu\_inv}, \sigma^*_{pago}\}$
- $\Psi_{Cestancia} = \{\sigma'_{beca}, \sigma'_{uo\_uni}, \sigma'_{obu\_uni}, \sigma'_{uo\_res}, \sigma'_{obu\_res}, \sigma'_{uo\_inv}, \sigma'_{obu\_inv}, \sigma'_{pago}\}$

4.1.1. Subtransacciones

En *SeRes* existen requisitos transaccionales que deben ser cumplidos. Por ejemplo, existe dependencia entre la residencia universitaria y el sistema de investigación: ambos deben pertenecer a la misma universidad. También existen relaciones de reemplazo ya que tanto UO como OBU pueden aceptar una propuesta. Estas relaciones se muestran en la **figura 3**.

s_{beca}: Recibe los datos del estudiante, duración de la estancia y los detalles de la universidad destino. Si la beca es concedida, contacta con *Pago* para transferir el dinero a la cuenta del estudiante. Responde con el resultado de la solicitud.

s_{uo_uni}, s_{obu_uni}: Reciben la propuesta del estudiante y responden si es aceptada o no.

s_{uo_res} , s_{obu_res}: Reciben la aceptación de la universidad, fechas de reserva y los datos del estudiante. Se reserva una habitación si hay disponible. Notifica a *Pago* la cuantía. Responde con los detalles de la reserva.

s_{uo_inv}, s_{obu_inv}: Reciben los datos del estudiante y fechas de la estancia. Registra al estudiante en el sistema y responde con el resultado.

s_{pago}:: Recibe los detalles de la cuenta y la cuantía ingresada o adeudada. Responde con el resultado.

4.1.2. Acciones compensatorias

c_{beca}: Cancela la beca otorgada e intenta concedérsela a otro estudiante.

c_{uo_uni} , c_{obu_uni}: Cancela la aceptación del estudiante.

c_{uo_res} , c_{obu_res}: Cancela la reserva y notifica a *Pago*.

c_{uo_inv} , c_{obu_inv}: Da de baja al estudiante del sistema.

c_{pago}: *Pago* cancela la operación.

4.1.3. Estados iniciales

σ_{beca}: El servicio debe haber recibido los datos del estudiante, duración de la estancia y la información de la universidad destino.

σ_{uo_uni} , σ_{obu_uni}: Debe haber recibido la propuesta del estudiante.

σ_{uo_res} , σ_{obu_res}: Debe haber recibido los datos personales del estudiante, fechas de la reserva y la aceptación de la universidad.

σ_{uo_inv} , σ_{obu_inv}: Debe haber recibido los datos personales del estudiante, fechas de la estancia y la aceptación de la universidad.

σ^*_{pago}: La cuantía de la beca debe haber sido depositada en la cuenta del estudiante. El pago de la residencia de estudiantes ha sido cargado en la misma cuenta.

4.1.4. Estados ejecutados

σ^*_{beca}: El servicio debe haber notificado la decisión. Si fue concedida, ha notificado *Pago* para la transferencia del dinero.

$\sigma^*_{uo\_uni}$, $\sigma^*_{obu\_uni}$: Debe haber recibido la propuesta del estudiante. Se ha almacenado en el sistema y se ha respondido con la decisión tomada.

$\sigma^*_{uo\_res}$, $\sigma^*_{obu\_res}$: Debe haber notificado el resultado sobre la reserva. Si había habitaciones disponibles, se registra la reserva acorde con las fechas recibidas.

$\sigma^*_{uo\_inv}$, $\sigma^*_{obu\_inv}$: El estudiante debe haber sido dado de alta en el sistema. Se ha creado un perfil y una cuenta de correo usando sus datos personales. Ha notificado el resultado.

σ^*_{pago}: La cuantía de la beca ha sido ingresada en la cuenta del estudiante. El pago de la reserva del alojamiento ha sido cargado en la misma cuenta.

4.1.5. Estados compensados

σ'_{beca}: La beca debe haber sido registrada como cancelada en el sistema de becas. Si existe algún estudiante esperando por una y el plazo está aun abierto, la aplicación ha comenzado un nuevo proceso de otorgación de beca.

σ'_{uo_uni} , σ'_{obu_uni}: El estudiante aceptado debe haber sido borrado del sistema. Si había algún estudiante rechazado, se puede reabrir algún proceso para una nueva aceptación.

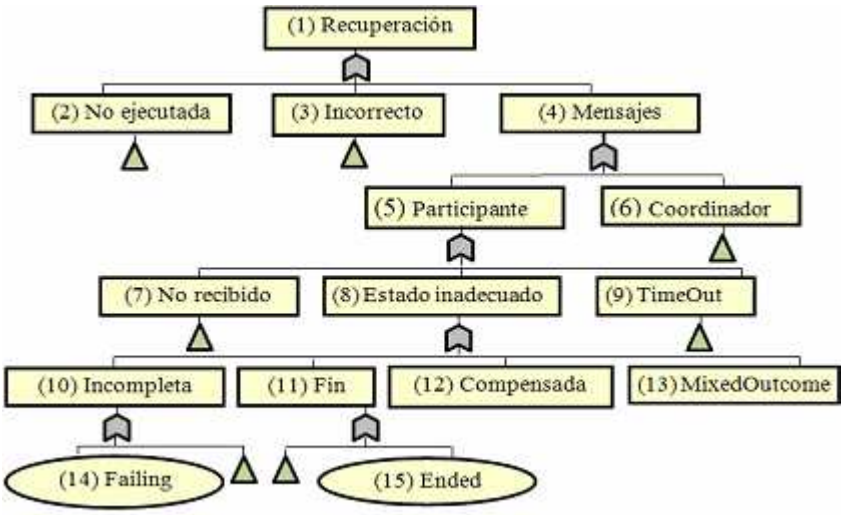

Figura 4. Árbol de fallos.

σ'_{uo_res}, σ'_{obu_res}: La reserva de la habitación debe haber sido cancelada y aparecer como disponible en el sistema. *Pago* debe haber sido notificado.
σ'_{uo_inv}, σ'_{obu_inv}: El usuario debe haber sido dado de baja en el sistema, quedando anulado su perfil y correo.
σ'_{pago}: Si el cobro del alojamiento no había sido cargado aun, debe haber sido anulado. En otro cado, el dinero debe haber sido reembolsado. Si el dinero ya había sido ingresado, el servicio debe haber devuelto esa cuantía.

5. Especificación de casos de prueba para la propiedad de recuperación

Se define riesgo como un evento indeseado que representa una amenaza para el correcto funcionamiento de un proceso [4]. El análisis de riesgos es una técnica usada para investigar problemas y proponer medios para mitigar sus daños. Aunque originalmente fue usada en industrias como la nuclear o espacial, actualmente también es usada en el desarrollo de software donde la seguridad puede ser muy importante [3]. Nuestro enfoque usa FTA para derivar jerárquicamente escenarios de riesgo durante el ciclo de vida de una *wT*. Cada *propiedad del sistema* es la raíz de un árbol de riesgos que se elabora determinando las posibles causas hasta alcanzar escenarios individuales [9].

La **figura 4** ilustra una parte del árbol de riesgos para la propiedad *Recuperación*. El árbol comienza con el riesgo principal (fallo en el proceso de compensación) que puede ocurrir debido a diferentes causas: la acción compensatoria no se ha ejecutado (2), se ha ejecutado de forma incorrecta (3), o hubo un problema con los mensajes de compensación (4). Para ilustrar este trabajo extendemos la rama derivada del nodo (4) hasta alcanzar nodos hoja que serán usados en los siguientes apartados para definir casos de prueba.

El riesgo (4) puede ser debido a problemas con los mensajes del participante (5) o del coordinador (6). Los problemas con el participante son: que no reciba el mensaje de recuperación (7), lo reciba cuando no debería (8) o que se reciba con el *timeout* cumplido (9). Cuando un participante tiene problemas al recibir los mensajes de compensación puede ser a que lo recibió estando en un estado inadecuado y ejecutando erróneamente la acción compensatoria. Significa que el participante recibe el mensaje sin haber ejecutado su sub-transacción (10), el participante lo recibe y ya ha abandonado su participación en la transacción (11), lo recibe cuando ya ha ejecutado la compensación (12) o lo recibe cuando no es necesario ejecutar ninguna (13).

Una de las posibles situaciones identificadas en (10) es que el participante esté en el estado *failing* (14). Esta situación significa que el participante mientras estaba ejecutando su subtransacción, alcanzó el estado *failing* debido a la aparición de un fallo.

Una posible situación basada en (11) es que el mensaje de compensación se recibe tras la confirmación de que la ejecución de su subtransacción fue correcta, estando, por tanto, en el estado *ended*. El resto de nodos serían desarrollados de la misma manera.

Los nodos hojas representan situaciones de posible riesgo que deberían ser probadas. Siguiendo la derivación jerárquica de las causas se pueden definir fácilmente especificaciones de casos de prueba para aplicaciones que usan WS-BA. En los siguientes apartados se usan los nodos hojas (14) y (15) para definir casos de prueba para *SerEs*.

5.1. Escenario *failing*

Este escenario de riesgo se refiere a un participante que ha recibido el mensaje de compensación cuando está en el estado *failing* (nodo hoja 14). Significa que ocurrió un problema mientras estaba ejecutando su subtran-sacción, por lo que su acción pudo no haber terminado completamente. Si ejecuta la acción compensatoria ocurriría que algunas características de la subtransacción se desharían cuando realmente no habían sido realizadas.

En el ejemplo de *SerEs* se ve la necesidad de comprobar esta situación. La **figura 5** presenta el caso de prueba acorde a este riesgo para evitar que un nuevo estudiante sea aceptado cuando no es posible. Imaginemos que el servicio *UO_uni* notifica un fallo mientras estaba estudiando una propuesta pero, indistintamente, no hay plazas disponibles para nuevos estudiantes. El estudiante original no fue aceptado pero debido a la acción compensatoria ejecutada tras la recepción del mensaje, el servicio notifica a un nuevo estudiante su aceptación. Este nuevo estudiante podría rechazar otras universidades creyendo erróneamente que tenía plaza en esta universidad.

5.2. Escenario *ended*

Este escenario se refiere a un participante que ha finalizado correctamente su transacción

Precondiciones	$wT_{estancia}$ está correctamente inicializada.
Secuencia de entrada	$UO\_universidad_{estancia}$ [register]K - $OBU\_universidad_{estancia}$ [register]K - K[complete] $UO\_universidad_{estancia}$ - $UO\_universidad_{beca}$ [fail]K - K[compensate] $UO\_universidad_{estancia}$
Salida esperada	*UO_universidad* ignora el mensaje de compensación por lo que no se ejecuta c_{uo_uni} Por tanto el sistema permanece en σ_{no_uni} y no se cumple $\sigma^*_{uo\_uni}$ *UO_universdad* queda a la espera del mensaje *failed*.

Figura 5. Caso de prueba para SerEs según el escenario failing.

Precondiciones	$wT_{estancia}$ está correctamente inicializada. Se cumple el estado $\sigma^{*}_{obu\_inv}$
Secuencia de entrada	$OBU\_inv_{estancia}[register]K$ - $K[complete]\ OBU\_inv_{estancia}$ - $OBU\_inv_{estancia}\ [completed]K$ - $K[ended]\ OBU\_inv_{estancia}$ - $K[compensate]\ OBU\_inv_{estancia}$
Salida esperada	$OBU\_inv_{estancia}$ ignora el mensaje de compensación por lo que no se ejecuta c_{obu_inv}. Por tanto el sistema permanece en $\sigma^{*}_{obu\_inv}$ y no se cumple σ'_{obu_inv}

Figura 6. Caso de prueba para *SerEs* según el escenario *ended*.

(nodo hoja 15). Por tanto recibe el mensaje de compensación cuando está en el estado *ended*. Podría ocurrir debido a un problema del coordinador, pero el participante debe estar preparado para ignorar este mensaje ya que si lo acepta, la subtransacción sería compensada. La **figura 6** muestra un caso de prueba para *SerEs* siguiendo este escenario.

Este fallo podría causar que una vez la transacción haya finalizado, el servicio *OBU_inv* borre de su sistema al estudiante. Dado que el participante está en el estado *ended*, el protocolo no permitirá notificar esta acción así que aparentemente la transacción se completó correctamente. Si *OBU_inv* ejecuta la acción compensatoria, se borrará el perfil de usuario perdiendo sus correos electrónicos, documentos de trabajo, etc.

6. Trabajos relacionados

La mayoría de trabajos abordan el modelado de transacciones desde una perspectiva de diseño [1][6]. Chalin [10] usa casos de usos para definir los requisitos transaccionales, mientras que Banagala [2] propone usar *Problem Frames Approach*.

A diferencia de esos trabajos, nosotros proponemos un sencillo modelo para la definición de requisitos transaccionales en una transacción WS-BA, facilitando además la derivación de los casos de pruebas.

No se han encontrado trabajos específicos para la prueba de aplicaciones basadas en WS-BA. Algunos esfuerzos existentes están centrados en la verificación formal. Lanotte [16] desarrolló un modelo para describir transacciones de larga ejecución y la verificación automática de propiedades usando *model checking*. Emmi [12] propone traducir procesos con acciones compensatorias en un *árbol autσmata* para verificar la ilusión de atomicidad. Li [17] presenta un modelo para la verificación de requisitos de atomicidad relajada usando *restricciones temporales*.

Este trabajo estudia la viabilidad de un enfoque práctico en contraste con otras investigaciones donde se usan modelos teóricos para una verificación formal. Nuestro enfoque se basa en la identificación de escenarios de riesgo donde se deben aplicar pruebas dinámicas, siendo el objetivo la especificación de casos de prueba para el estándar WS-BA.

7. Conclusiones

Hemos presentado un enfoque práctico para la prueba de la *recuperabilidad* en transacciones según el estándar WS-BA. Este trabajo propone un sencillo modelo para la especificación de loa requisitos funcionales. Los casos de prueba se definen siguiendo los escenarios alcanzados tras un análisis de riesgo usando FTA. La viabilidad de este enfoque se ilustra con un caso de estudio. Dado que las transacciones se basan en servicios independientes, el objetivo de esta línea de investigación es mejorar las pruebas en la fase de integración. Se necesita más investigación para analizar la relación y complementariedad de este enfoque con otras técnicas.

Un trabajo a corto plazo es ejecutar los casos de prueba definidos en una implementación [15] para medir su calidad. Se trabajará también en la mejora del modelo para hacerlo independiente del estándar utilizado. El objetivo es definir un modelo abstracto que incluya la definición de los requisitos así como los riesgos identificados, facilitando la generación de casos de prueba.

Agradecimientos

Este trabajo fue parcialmente financiado por el Ministerio de Ciencia y Tecnología, España, Programa Nacional I+D+I, proyecto Test4DBS (TIN2010-20057-C03-01) y por la beca BES-2008-004355.

Referencias

[1] M. Alrifai, P. Dolog,W. Nejdl. Transactions Concurrency Control in Web Service Environment. *ECOWS, 2006*.
[2] V. Banagala. Analysis of transaction problems using the problem frames approach. *IWAAPF, 2006*.
[3] J. Bennett, G. Bohoris, E. Aspinwall, R. Hall. Risk analysis techniques and their application to software development. *EJOR, 1996*, *95*, pp. 467-475.
[4] A. Benoit, M. Patri, S. Rivard. A framework for information technology outsourcing risk management. *ACM SIGMIS Database, 2005, 36*, pp. 9-28.
[5] OASIS. *Business Transaction Protocol: An OASIS Committee Specification*, Version 1.0, June 2002, <http://www.oasis-open.org/committees/download.php/1184/2002-06-03.BTP_cttee_spec_1.0.pdf >.
[6] M. Butler, C. Ferreira , M. Ng. Precise modeling of transactions and its application to BPEL. *Journal of Universal Computer Science, 11, 2005*.
[7] G. Canfora, M. Di Penta. Service-Oriented architectures testing: a survey. *LNCS, 2009, 5413*, pp. 78-105.
[8] R. Casado, J. Tuya. "Testing transactions in service oriented architectures", *ICWE, DC, 2009.*
[9] R. Casado, J. Tuya, M. Younas. Testing long-lived web services transactions using a risk-based approach. *QSIC, 2010*.
[10] P. Chalin, D. Sinning, K. Torkzadeh. Capturing business transaction requirements in use case models. *ACM SAC*, *2008*.
[11] F. Curbera, R. Khalaf, N. Mukhi, S. Tai, S. Weerawarana. The next step in web services. *Communications of the ACM, 2003, 46*, pp. 29-34.
[12] M. Emmi, R. Majumdar. Verifying compensating transactions. *LNCS, 2007, 4349*, pp. 29-43.
[13] H. Garcia–Molina, K. Salem. Sagas. *ACM SIGMOD, 1987, 16*, pp. 249-259.
[14] C. Guidi, R. Lucchi, M. Mazzara. A formal framework for web services coordination. *Electronic Notes in Theoretical Computer Science, 2007, 180*, pp. 55-70.
[15] JBoss Community. <http://www.jboss.org/jbosstm>.
[16] R. Lanotte, A. Maggiolo-Schettini, P. Milazzo, A. Troina. Design and verification of long-running transactions in a time framework. *Science of Computer Programming, 2008, 73*, pp. 76-94.
[17] J. Li, H. Zhu, J. He. Specifying and verifying web transactions. *LNCS, 2008, 5048*, pp. 149-168.
[18] **W. Vesley, F. Goldberg, N. Roberts, D. Haasl.** Fault tree handbook. *NUREG-0492, U.S. Nuclear Regulatory Commission*, 1981.
[19] OASIS. *Web Service Atomic Transactions*, <http://docs.oasis-open.org/ws-tx/wstx-wsat-1.2-spec-os/wstx-wsat-1.2-spec-os.html>.
[20] OASIS. *Web Service Business Activity*, <http://docs.oasis-open.org/ws-tx/wstx-wsba-1.2-spec-os/wstx-wsba-1.2-spec-os.html>.
[21] OASIS. *Web Services Composite Application Framework*, <http://docs.oasis-open.org/ws-caf/ws-context/v1.0/OS/wsctx.html>.
[22] OASIS. *Web Service Coordination*, <http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.2-spec-os/wstx-wscoor-1.2-spec-os.html>.
[23] OASIS. *WS-BPEL*, <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>.
[24] M. Younas, K. Chao, C. Lo, Y. Li. An efficient transaction commit protocol for composite web services. *AINA, 2006, 1*, pp. 591-596.