Una Arquitectura para el Diseño de Soluciones de Integración de Aplicaciones Empresariales con Soporte para Tolerancia a Fallos*

Rafael Z. Frantz
Dep. de Tecnología
Universidad UNIJUÍ
São Francisco, 501
Ijuí 98700-000 RS, Brasil
rzfrantz@unijui.edu.br

Rafael Corchuelo
Dep. de Lenguajes y Sist. Informáticos
Universidad de Sevilla
Avda. Reina Mercedes s/n
41012 Sevilla, España
corchu@us.es

Carlos Molina-Jiménez
School of Computing Science
University of Newcastle, UK
NE1 7RU
Newcastle upon Tyne, UK
carlos.molina@ncl.ac.uk

Resumen

Las soluciones de integración de aplicaciones empresariales (EAI) suelen estar basadas en workflows de mensajes gracias a los cuales es posible conseguir que dos o más aplicaciones cooperen para proporcionar un nuevo servicio o que mantengan sus datos sincronizados. La bibliografía proporciona resultados sobre dos modelos de ejecución diferentes: el *basado en procesos* y el *basado en tareas*. En cualquiera de los dos es común tener que tratar con fallos, debido a que las aplicaciones integradas suelen estar totalmente distribuidas y en rara ocasión fueron diseñadas teniendo en cuenta que deberían ser integradas posteriormente. En este artículo, presentamos una arquitectura para proporcionar tolerancia a fallos en soluciones integración de aplicaciones empresariales que utilizan el modelo de ejecución basado en tareas.

1. Introducción

En las empresas actuales es habitual encontrar un conjunto bastante heterogéneo de aplicaciones (conocido como ecosistema software [20]) que regularmente incluye aplicaciones desarrolladas dentro de la empresa para solucionar un problema específico y aplicaciones compradas a varios proveedores. Las últimas incluyen aplicaciones relativamente nuevas y aplicaciones completamente desactualizadas.

Ejemplos de estas aplicaciones son los sistemas para procesamiento de nóminas y sistemas de venta. Con frecuencia, en estos ambientes surge la necesidad de integrar las aplicaciones del ecosistema software para mantener sincronizados los datos que ellas manipulan o bien para crear nuevos servicios a partir de los servicios que las aplicaciones del ecosistema ofrecen [13]. Este campo de investigación es conocido como Integración de Aplicaciones Empresariales (EAI). El reto de las soluciones EAI consiste en diseñar e implementar un conjunto de *procesos de wrapping* que se encargan de interactuar con las aplicaciones (un wrapper para cada aplicación) y un conjunto de *procesos de integración*, que se encargan de gestionar el flujo de mensajes entre las aplicaciones.

*Se proporciona material adicional en la dirección http://www.tdg-seville.info/rzfrantz/JSS-2010: una implementación del sistema, código fuente, documentación y videos con demostraciones. El primer autor llevó a cabo parte de su trabajo en la Universidad de Newcastle (Reino Unido), durante una estancia de investigación. Su trabajo fue financiado por el Evangelischer Entwicklungsdienst e.V. (EED). El segundo y el primer autor han sido parcialmente financiados por la Comisión Europea (FEDER), los programas de I+D+I del MEC Español y la Junta de Andalucía (proyectos TIN2007-64119, P07-TIC-2602, P08-TIC-4100, y TIN2008-04718-E) y por el programa de investigación de la Universidad de Sevilla. El tercer autor fue financiado parcialmente por el Gobierno de Reino Unido, a través del proyecto EPSRC No. EP/D037743/1.

Una buena alternativa para soportar el diseño de los procesos de integración son los Sistemas de Soporte a Procesos (PSS): un middleware que, entre otras facilidades, ofrece recursos para diseñar procesos distribuidos (e.g., soluciones EAI) y para monitorizar su ejecución [12]. Ejemplos representativos de PSSs son los sistemas de workflow tradicionales [12, 1]. En [26] se estudian los PSSs basados en BPEL [23]. Ejemplos de PSSs enfocados a soluciones EAI son BizTalk [8], Tibco [25], Camel [10] y Mule [22].

Una metodología muy conocida en el diseño de soluciones EAI consiste en usar una base de datos para almacenar los mensajes entrantes (mensajes procedentes de las aplicaciones individuales y dirigidos hacia el proceso de integración) uno por uno hasta que todos los mensajes necesarios para activar el proceso de integración hayan llegado. En ese momento se solicita y activa un hilo (*thread*) solo que comienza a ejecutar, de forma síncrona, todas las tareas del proceso. Este modelo de ejecución se conoce como el *basado en procesos*. Se debe observar que, si en algún momento una tarea envía un mensaje al wrapper de una aplicación, el hilo queda bloqueado hasta que la aplicación responde. Con el propósito de minimizar interferencias, es frecuente que en entornos de integración de aplicaciones, las aplicaciones den poca prioridad a los mensajes que reciben de los procesos de integración. Es común también que para responder a un mensaje, la aplicación necesite de la intervención de una persona. En otros casos, la aplicación que recibe el mensaje no es una aplicación simple, sino que también es una solución de integración. Debido a estas posibilidades, la respuesta puede demorarse horas o incluso días [12]. Consecuentemente, uno de los riesgos del modelo de ejecución basado en procesos es que el hilo asignado puede permanecer largos períodos inactivo y consumiendo vanamente recursos.

Para intentar paliar este problema, algunos autores han propuesto técnicas de deshidratación/rehidratación de hilos [8, 26] que consisten en serializar y guardar en disco los procesos que al estar en espera de mensajes de respuestas, mantienen bloqueados hilos durante mucho tiempo. Cuando las respuestas llegan, el proceso se recupera, se deserializa y se ejecuta en cuanto hay un hilo disponible. El inconveniente de esta estrategia, es el coste de la deshidratación/rehidratación.

Otra alternativa es usar un modelo de ejecución *basado en tareas*. En este modelo, los hilos no se asignan a los procesos, sino a las tareas que procesan los mensajes. Al disminuir el nivel de granularidad, se consigue mayor concurrencia y aprovechamiento de recursos [15, 11]. Por ejemplo, los mensajes comienzan a procesarse (algunos concurrentemente) en cuanto son recibidos; de su correlación se encargan tareas específicas dentro de los procesos de integración.

Merece la pena recordar que las soluciones EAI son intrínsecamente distribuidas y por lo tanto son susceptibles a una amplia gama de fallos [12, 24]. Esto se debe a que las EAI involucran varias aplicaciones comunicadas a través de una red y a que tanto las aplicaciones como la red son susceptibles a fallos. La red por ejemplo, puede imprevisiblemente retrasar, corromper e incluso perder mensajes. Para ser de utilidad práctica, las soluciones EAI deben incluir mecanismos de tolerancia a fallos [4].

La tolerancia a fallos en el contexto de los PSS no es un tema completamente nuevo; varios autores ya han estudiado el problema, pero únicamente dentro del contexto del modelo de ejecución basado en procesos que hemos descrito anteriormente y considerando la existencia de un proceso solo; es decir, han dejado fuera de sus análisis todas las soluciones EAI que se componen de varios procesos de integración y de wrapping. Con el propósito de aportar una solución a este problema, en este artículo presentamos una arquitectura para proporcionar tolerancia a fallos a soluciones EAI, que utilizan el modelo de ejecución basado en tareas. La arquitectura funciona bajo dos condiciones alcanzables en la práctica: a) que podemos enriquecer los mensajes producidos por los procesos de integración y wrapping con información obtenida de los mensajes usados para producirlos. b)

que el monitor que usamos para detectar fallos (Figura 3) está exento de fallos; este tema queda fuera del alcance del presente trabajo; nos limitamos a mencionar que este requisito se puede cubrir con la ayuda de técnicas de implementación con versiones múltiples [3].

El artículo se organiza de la siguiente manera: en la Sección 2 analizamos trabajos relacionados con el tema; la Sección 3, introduce los conceptos necesarios para comprender el tema. La arquitectura que proponemos se presenta en la Sección 4; la Sección 5, describe el estudio y solución a un problema real que ayuda a entender la propuesta; finalizamos presentando nuestras conclusiones en la Sección 6.

2. Trabajos Relacionados

Nuestro estudio sobre la tolerancia a fallos en soluciones EAI está directamente relacionada con el concepto de transacciones de larga duración introducido en [21]. Este trabajo de investigación inspiró y fomentó otros trabajos dirigidos a proveer los PSS con mecanismos de tolerancia a fallos. El trabajo original ha dado lugar a varias ideas para solucionar el problema, que si omitimos los detalles, podemos clasificar en dos grupos: las que se basan en algoritmos para reaccionar a fallos de forma automática y las que se basan en reglas, en donde los fallos son tratados por las reglas definidas por el diseñador. En [6] se estudia un modelo abstracto para workflows con tolerancia a fallos; este artículo también proporciona una buena revisión de las metodologías que existen para enfrentar el problema. En [4] se presenta un algoritmo para implementar transacciones en aplicaciones distribuidas, donde las aplicaciones participantes cooperan entre ellas para implementar un protocolo distribuido.

En [16] se presenta un algoritmo para la ejecución de procesos BPEL con transacciones relajadas, donde se relaja la propiedad de *atomicidad*. En [17] los autores mejoran sus resultados anteriores introduciendo un lenguaje basado en reglas que permite la definición de reglas específicas de recuperación de fallos. La solución descrita en [7] propone diseñar y gestionar procesos especiales para aportar tolerancia a fallos a un workflow orquestado con BPEL. En dicha propuesta, los procesos del workflow que pueden fallar llevan asociados a ellos un proceso especial con la lógica para la recuperación en caso de que ocurra un fallo. Cuando el sistema detecta un fallo en alguno de los procesos del workflow, activa el proceso especial correspondiente. En [12] se propone un algoritmo para el tratamiento de fallos en aplicaciones integradas por medio de sistemas de workflow y tratables con acciones de compensación o con el protocolo de commit en dos fases. [18] presenta un mecanismo para el tratamiento de fallos en aplicaciones en las que las acciones de compensación son difíciles o imposibles de llevar a la práctica.

En [5] se propone una arquitectura para implementar tolerancia a fallos por medio de workflows ad-hoc. Por ejemplo, utilizando un servidor web como front-end, un servidor de aplicaciones, una base de datos y un sistema de log. Los autores asumen que un workflow se activa siempre que le llega un mensaje de petición que fluirá a través de los componentes del workflow; por lo tanto, su mecanismo de recuperación de fallos se basa en el seguimiento de todos los mensajes a través del sistema. De forma parecida, en nuestra propuesta nosotros hacemos un seguimiento de mensajes para conocer toda la traza de un mensaje durante su ejecución en el workflow de integración.

En [19] el autor presenta ideas para proporcionar tolerancia a fallos a procesos de workflow que manejan peticiones asíncronas. También presenta una buena discusión respecto al modelo de comunicación asíncrono entre procesos. Dicha propuesta se centra en el modelo de ejecución basado en procesos y por consiguiente, padece de las desventajas que explicamos en la Sección 1; además, asume que la interface de comunicación entre los procesos es un mecanismo que permite implementar una cola de mensajes. En la arquitectura que estamos presentando, nosotros consideramos un modelo de ejecución basado en tareas y

sin asumir ninguna implementación específica del mecanismo de comunicación de los procesos. Dicho mecanismo puede ser una interface gráfica, una API de programación, una cola u otro. El autor también comenta que con ese mecanismo de comunicación basado en colas, además del uso de las técnicas de replicación y balanceo de carga para los procesos, se puede aportar tolerancia a fallos a procesos de workflow.

En [1] los autores proponen el uso del mecanismo de manejo de excepciones para añadir tolerancia a fallos a soluciones de workflow, pero combinada con el uso de estrategias de replicación para aumentar la disponibilidad de las soluciones. En [9] se describe una arquitectura para la tolerancia a fallos basada en una máquina de estados finitos (diagramas de secuencia de mensajes) y que reconoce secuencias válidas de mensajes en los workflows. Las acciones de recuperación se activan cuando se encuentra un mensaje inválido o el tiempo de ejecución de la maquina de estados se agota.

En [15] y [11] se presentan una solución para proporcionar tolerancia a fallos en procesos basados en redes de Petri. El controlador original se incorpora sin ningún cambio en un nuevo controlador que conserva la estructura del controlador original pero enriquecido con lugares adicionales, transiciones y fichas, con el objetivo de detectar los fallos.

Después de estudiar los trabajos citados anteriormente, llegamos a la conclusión de que coinciden en varios puntos: Prácticamente todos atacan el problema de la tolerancia a fallos considerando que las soluciones EAI están compuestas por proceso solo, con excepción de [19], que se centra en la comunicación asíncrona entre procesos. Además, dichas propuestas asumen que los procesos ejecutan un conjunto de mensajes previamente correlacionados; los procesos de la propuesta [5] tampoco funcionan con múltiples mensajes entrantes. Esto es una deficiencia importante pues las soluciones EAI normalmente suelen componerse de varios procesos de integración y de wrapping; también, la incapacidad para hacer frente a múltiples mensajes entrantes es importante puesto que hay muchos casos en que un proceso de integración se debe iniciar con múltiples mensajes procedentes de distintas aplicaciones. Otra limitación común entre los trabajos que hemos revisado es que, exceptuando [11, 15], todos están pensados para el modelo de ejecución basado en procesos. Además, los trabajos presentados en [11, 15] están enfocados a procesos de control industrial representados como redes de Petri, naturalmente, no consideran los problemas causados por la naturaleza distribuida de las soluciones EAI, ni los que introducen los ecosistemas software, ni otros problemas intrínsecos de las soluciones EAI.

3. Conceptos fundamentales

En esta sección presentamos los conceptos fundamentales del contexto de integración de aplicaciones empresariales en que trabajamos, además introducimos la semántica de fallos atacada por nuestra propuesta.

3.1. Soluciones EAI

La Figura 1 muestra una solución típica de EAI que contiene tres aplicaciones (App1, App2 y App3) y dos procesos de integración que implementan la lógica de negocio de la integración. Consecuentemente, contiene también tres procesos de wrapping que se usan para comunicar la solución con las tres aplicaciones por un lado y con dos procesos de integración por el otro. Los dos procesos de integración usan puertos conectados por canales de comunicación para comunicarse entre ellos y con las aplicaciones. Los puertos encapsulan tareas que se encargan de recibir, hacer peticiones, enviar y de otras actividades similares. La encapsulación en los puertos abstrae y esconde los detalles del mecanismo de comunicación. Detrás de la abstracción podría haber desde un protocolo basado en RPC funcionando sobre HTTP hasta un protocolo basado en documentos e implementado sobre un sistema de gestión de base de datos.

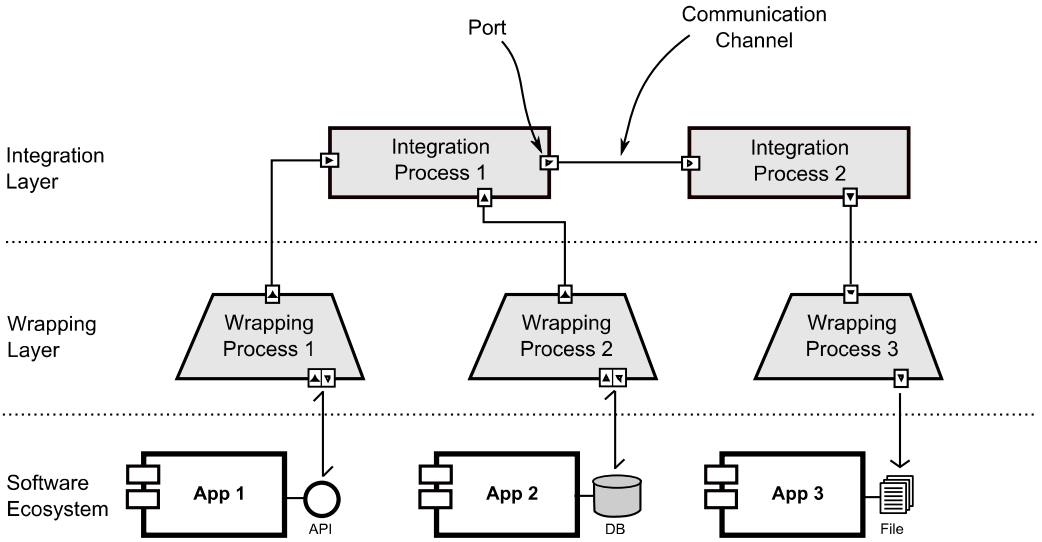


Figura 1: Capas de una solución EAI típica.

3.2. Semántica de fallos

De manera informal, se dice que un sistema es confiable si sus usuarios (humanos u otros sistemas) se fían del servicio que el sistema ofrece. La tolerancia a fallos es un requisito fundamental para ofrecer confiabilidad. Su estudio se basa en tres conceptos básicos: defectos, errores y fallos. Estos conceptos los explican ampliamente en [14, 2], sin embargo, los explicaremos aquí brevemente. Un *defecto* es un daño (desperfecto) que existe en un sistema, por ejemplo, una línea de código incorrecta y que tarde o temprano puede ser activado. Cuando un defecto es activado, causa un error. Un *error* es la manifestación de un defecto y se percibe dentro del sistema como una desviación de su comportamiento normal. Si no es tratado, un error puede causar otros errores y propagarse hasta alcanzar la interface del usuario. Cuando la alcanza, el usuario dice que el sistema ha fallado o que a su sistema le ha ocurrido un fallo. Es decir, un *fallo* es la percepción por parte del usuario de un comportamiento incorrecto del sistema.

Regresando al tema de las soluciones EAI, vale la pena no perder de vista que estas son sistemas compuestos de otros sistemas. Por lo tanto, el diseñador debe de tomar en cuenta que tanto la solución EAI como las aplicaciones que componen el ecosistema software pueden, tarde o temprano, fallar. Vulnerables a fallos son los procesos, puertos y canales de comunicación. Para ofrecer soluciones EAI con mecanismos de tolerancia a fallos, el diseñador necesita primero identificar la semántica de los fallos de los componentes. Brevemente, la semántica de fallos es la enumeración de los comportamientos anormales del sistema, que el diseñador conoce y espera que tarde o temprano alcancen la interface del usuario. Después de identificar la semántica de fallos, el diseñador debe estipular que tipos de errores la solución EAI debe de tolerar. Los errores se toleran evitando que estos lleguen hasta la interface del usuario. La técnica mas conocida consiste en: 1) detectar el error durante la ejecución y 2) ejecutar una acción de compensación para, de ser posible, revertir el efecto del error y regresar al sistema a un estado normal. Nuestra arquitectura considera los siguientes tipos de fallos: de omisión, de respuesta, de tiempo y de proce-

samiento de mensajes.

Fallos de Omisión (OMF): Asumimos que una vez que una operación de comunicación se inicia por un puerto, ésta termina dentro de un intervalo de tiempo estrictamente definidos y reporta una de dos: éxito o fracaso. Los fallos de omisión modelan situaciones en donde problemas de red, de aplicaciones y de canales de comunicación, impiden que los puertos envíen o reciban un dato dentro del intervalo de tiempo definido.

Fallos de Respuesta (REF): Ocurren cuando una aplicación o canal de comunicación envía mensajes incorrectos a la solución. Esta los detecta aplicando un examen de validación (e.g., a la cabecera y cuerpo del mensaje) cuyo resultado es una de dos: éxito o fallo. Si el resultado es fallo, el mensaje no es procesado.

Fallos de Tiempo (TMF): Suelen ocurrir cuando un mensaje tiene un plazo para llegar al final del flujo. La verificación del plazo la hacen los puertos al analizar el mensaje. Un puerto notifica éxito cuando el plazo no se ha vencido y fallo en caso contrario. En gran parte, los TMF dependen de los fallos que ocurren dentro de la solución y de los que ocurren dentro del ecosistema software.

Fallos de Procesamiento de Mensajes (MPF): Los puertos y procesos notifican MPF cuando algo les impide completar el procesamiento de un mensaje; de lo contrario, notifican éxito.

Vale la pena aclarar que los OMF y REF son fallos debido a errores que ocurren dentro de las aplicaciones o en el medio de comunicación entre las aplicaciones y el proceso de integración. El diseñador asume que estos errores no son tratados (ni dentro de la aplicación, ni en la línea de comunicación, respectivamente) y que alcanzaran la interface entre la aplicación y el proceso de integración de la solución EAI. El objetivo de tratarlos, es evitar que estos fallos se propaguen hasta las aplicaciones. Los fallos TMF y MPF ocurren dentro del proceso de solución EAI.

4. Arquitectura propuesta

Se introduce en la Figura 2, por medio de un metamodelo, la arquitectura que proponemos para ofrecer soluciones EAI con tolerancia a fallos. Este metamodelo contiene los componentes que se necesitan para la construcción de las reglas, patrones de intercambio y mecanismos para la detección y recuperación de fallos, o dicho de otra manera, para restablecer el sistema cuando es afectado por un fallo. De acuerdo con este metamodelo, una EAISolution puede definir múltiples ExchangePatterns (MEPs) y Rules. Los Events son notificaciones para el monitor y son generados por los Sources dentro de la solución EAI, que de acuerdo con nuestra semántica cuenta con EventTriggerMessageType para notificar éxito o fallo durante el proceso de ejecución del workflow. Una fuente puede ser un Port o un Process. Cada MEP define un conjunto de puertos fuente entrantes y salientes de los que se reportan eventos. De forma resumida, el MEP representa el comportamiento que se espera de la solución EAI, es decir, cuando solución EAI tiene éxito a la hora de procesar un conjunto de mensajes correlacionados. Una regla esta compuesta por una Condition y una Action. Una condición puede ser SimpleCondition que es representada por un sólo evento, o CompositeCondition que contiene dos condiciones conectadas por un LogicalOperator. Cuando la condición es *true*, Action ejecuta el correspondiente bloque de acción de recuperación definido por la regla. El Monitor observa una o más soluciones EAI para detectar posibles fallos y activar los mecanismos necesarios para tratarlos. Como se puede ver en la Figura 3, el monitor está compuesto por un Log, un SessionCorrelator, un ExchangeEngine, un RuleEngine, un EntryPort y varios ExitPorts. Se debe observar que la arquitectura propuesta se centra en proporcionar tolerancia a fallos a las soluciones EAI y no incluye dicho soporte a propio mecanismo de tolerancia a fallos, es decir, al monitor.

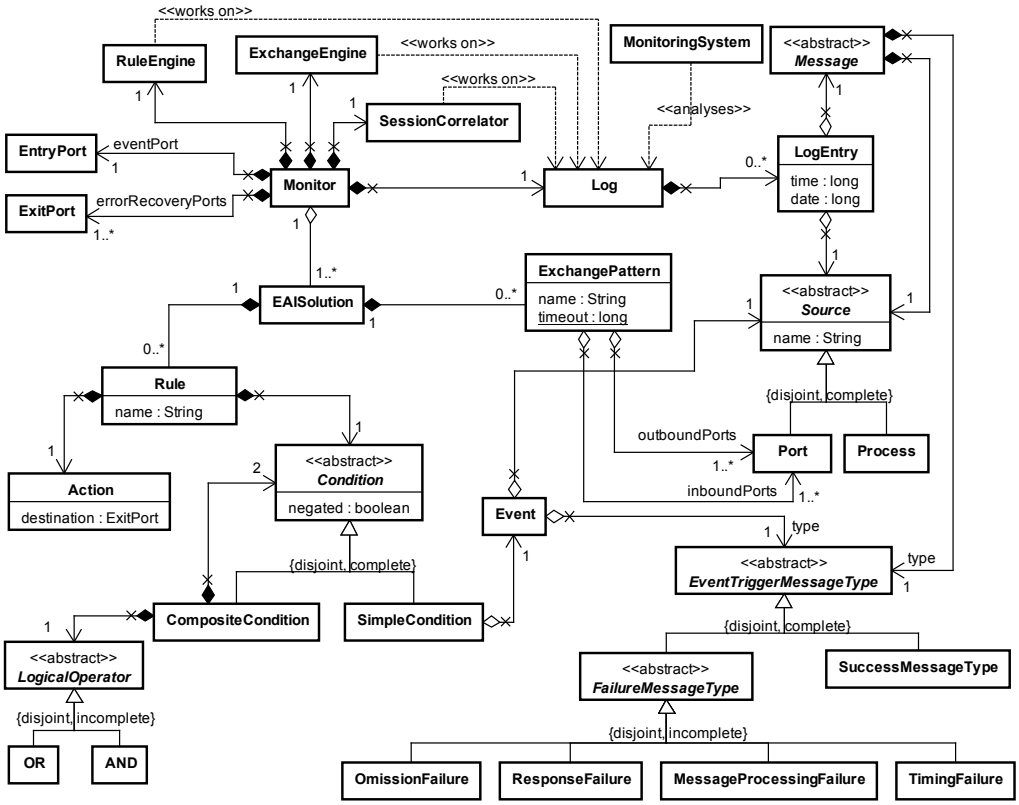


Figura 2: Metamodelo de una solución EAI con tolerancia a fallos.

4.1. El Sistema de Log

El sistema de log está representado por el elemento Log, donde se almacenan permanentemente todos los eventos de éxito y fallos reportados por las fuentes dentro de la solución EAI. El monitor recibe los eventos a través de un puerto de entrada y los almacena en el log; ahí permanecen disponibles para los demás componentes del monitor. En términos generales, el log está compuesto de varias LogEntries que contienen información sobre los eventos, tales como, el nombre completo de la fuente del evento, fecha y hora que ocurrencia y una copia del mensaje que se estaba procesando en el momento de la ocurrencia del evento. Por nombre completo de la fuente debemos entender, el nombre de la solución EAI que originó el evento, seguido por el nombre único de la fuente dentro de la solución. La unicidad del nombre de las fuentes de los eventos permite que el monitor observe una o más soluciones EAI simultáneamente. El log es compartido por el SessionCorrelator, el ExchangeEngine, y el RuleEngine. El log también puede proporcionar información a un MonitoringSystem, interesado en evaluar el estado de la solución EAI.

4.2. El Session Correlator

El Session Correlator busca en el log mensajes que están correlacionadas dentro de la misma sesión de workflow. Su salida es uti-

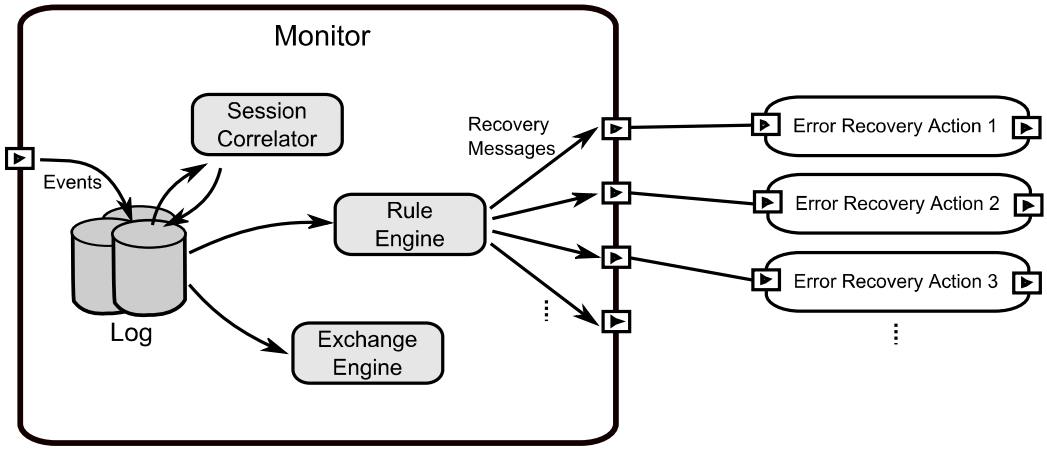


Figura 3: Vista abstracta del monitor.

lizada por el Exchange Engine para determinar el estado de las instancias de los MEPs y para desencadenar acciones de compensación. Dado que en nuestra arquitectura una tarea dentro de un proceso puede tomar varios mensajes de entrada y producir varios mensajes de salida, no es trivial determinar qué mensajes pertenecen a las diferentes sesiones de workflow. Para resolver el problema, enriquecemos los mensajes con información acerca de los mensajes originales utilizados para crearlos; después establecemos una relación padre-hijo entre mensajes para buscar su correlación dentro de la sesión: dos mensajes cualquiera m_a y m_b están correlacionados dentro de la misma sesión si m_a es el padre de m_b o m_b es el padre de m_a. Asimismo, tres mensajes m_a, m_b y m_c están correlacionados dentro de la misma sesión si m_c es un predecesor de ambos m_a y m_b. Además, un mensaje siempre esta correlacionado con el mismo.

4.3. El Exchange Engine

El Exchange Engine se encarga de gestionar los MEPs. Un MEP representa una secuencia de intercambio de mensajes entre las aplicaciones participantes. En la Figura 4 ilustramos la notación textual que usamos para especificar MEPs. Una solución EAI puede tener uno o más MEPs, por lo tanto, diferentes workflows de mensajes pueden ocurrir dentro de una solución EAI.

Los MEPs consideran sólo mensajes de éxito recibidos desde los puertos monitorizados en los conjuntos Inbound y Outbound, por lo tanto no hace falta que se especifique explícitamente los tipos. Cuando el Exchange Engine encuentra en el log dos o más mensajes que están correlacionados dentro de la misma sesión y que han venido de distintos puertos de un MEP, se crea una instancia de ese MEP y se le atribuye un *max-time-to-live*; max-time-to-live es un parámetro global y especifica un plazo límite para que la ejecución de una instancia termine. Los mensajes correlacionados dentro de la misma sesión pueden encajar en uno o más MEPs, si esto ocurre se crea una instancia de cada uno de los MEPs. Inbound tiene un conjunto de nombres completos de puertos, desde los que se originan los mensajes entrantes. De igual forma, Outbound tiene un conjunto de nombres de puertos a los que se envían mensajes de salida. La sintaxis para nombres completos es la siguiente: eai_solution_name::process_name::port_name, en que eai_solution_name por defecto corresponde a EAISolution.

```
EAI SOLUTION name
TIMEOUT time

EXCHANGE PATTERN name
  INBOUND (inbound_ports)
  OUTBOUND (outbound_ports)
              ⋮
RULE name
  CONDITION (condition)
  ACTION destination
              ⋮
```

Figura 4: Sintaxis de los Exchange Patterns y Reglas.

El Exchange Engine se encarga de detectar en las soluciones EAI los MEPs concluidos, los activos y los incompletos; también es capaz de detectar los mensajes que se encuentran en el log durante mucho tiempo y que no han sido tomados en cuenta para ningún MEP. A estos mensajes se conocen como *odd messages*. Un MEP concluido indica que se ha procesado con éxito en la solución EAI varios mensajes entrantes correlacionados dentro del plazo establecido en max-time-to-live. El Exchange Engine busca y detecta en el log todos los mensajes de salida que están correlacionados dentro de la misma sesión para una instancia de MEP. Una instancia de MEP activa contiene dos o más mensajes correlacionados (que no necesariamente son de salida) en el log; tiene un plazo establecido por max-time-to-live vigente y esta en espera de más mensajes para poder continuar su ejecución. Un MEP activo se considera incompleto cuando se vence el tiempo de su instancia. Las instancias de los MEPs pueden no completar debido a los fallos que pueden ocurrir durante la ejecución de la solución EAI. Cuando esto pasa, se activa el Rule Engine que cuando sea necesario, activa la ejecución de un bloque de acción de recuperación (mirar Figura 5).

Es posible que una instancia incompleta de MEP se complete después del vencimiento del plazo y después de la ejecución de un bloque de acción de recuperación. Estas situaciones son detectadas y notificadas por el Monitoring System.

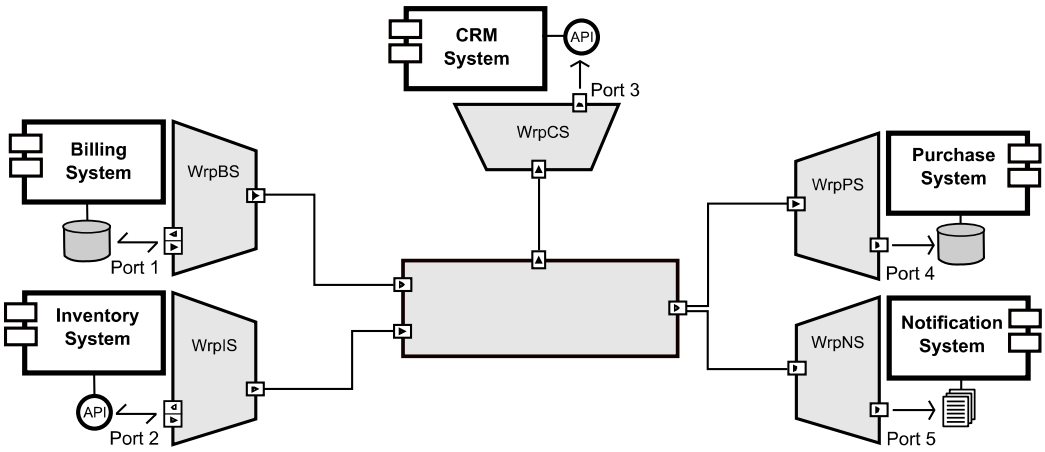
4.4. El Rule Engine

El Rule Engine se encarga de gestionar las reglas de Evento-Condición-Acción (ECA) de una solución EAI. Por ejemplo, cuando la condición de una regla arroja *true*, el Rule Engine crea un mensaje de recuperación y activa un bloque de acción de recuperación por medio de un puerto de salida. El bloque de acción de recuperación contiene lógica para tratar el error. Los bloques de acción de recuperación son externos al monitor y aunque han sido diseñados especialmente para funcionar con una aplicación y canal de comunicación particular, se pueden reciclar en otros ambientes.

Las reglas reaccionan tanto a los eventos de éxito como a los de fallos. Los eventos se producen tanto en los puertos como en los procesos, y llevan el nombre de la fuente y del tipo de evento, cf. Figura 2. La sintaxis general de nombres es la siguiente: eai_solution_name::source_name:event_type. Cuando el origen es un Puerto, es necesario incluir el nombre del proceso al que el puerto pertenece, cf. Figura 5.

5. Caso de Estudio

Para demostrar nuestra propuesta, presentamos en la Figura 5 una solución EAI (implementada con la tecnología Java) para una empresa ficticia. En esta solución se integran cinco aplicaciones ya existentes en la empresa que funcionan de forma independiente y que no fueron diseñadas para ser integradas.




```
EAI SOLUTION MySolution
TIMEOUT 24h

EXCHANGE PATTERN MyMEP_1
  INBOUND (WrpBS::Port1, WrpIS::Port2)
  OUTBOUND (WrpCS::Port3, WrpPS::Port4)

EXCHANGE PATTERN MyMEP_2
  INBOUND (WrpBS::Port1, WrpIS::Port2)
  OUTBOUND (WrpCS::Port3, WrpPS::Port4, WrpNS::Port5)
```

```
RULE MyRule_1
  CONDITION (!WrpPS::Port4:OK & WrpCS::Port3:OK)
  ACTION ErrorRecoveryActionForCS

RULE MyRule_2
  CONDITION (!WrpCS::Port3:OK & WrpPS::Port4:OK)
  ACTION ErrorRecoveryActionForPS

RULE MyRule_3
  CONDITION (WrpNS::Port5:OMF)
  ACTION ErrorRecoveryActionForCS
```

Figura 5: Ejemplo de una solución EAI con tolerancia a fallos.

La solución EAI tiene un solo proceso de integración y cinco procesos de wrapping, uno para cada aplicación. La función de esta solución es obtener facturas del Billing System (BS), mezclarlas con sus ordenes de pedidos correspondientes y proporcionadas por el Inventory System (IS), y producir un mensaje único con toda la información. Una copia de este mensaje se envía al CRM System (CS) y otra copia al Notification System (NS), que se encarga de notificar al cliente sobre su compra. Finalmente, se envía otra copia del mensaje al Purchase System (PS), que se encarga de registrar las compras. Una factura puede estar relacionada con varios pedidos; cuando esto ocurre se utiliza el número del pedido para hacer una correlación local.

Para ayudar a entender mejor algunas de las características de la arquitectura que proponemos, asumimos que la solución EAI nos impone tres restricciones. 1) Los mensajes que se van a mezclar se deben enviar con éxito al CS y al PS por medio de los puertos Port 3 y Port 4, respectivamente. Cualquier fallo que impida a una de las aplicaciones recibir el mensaje que está correlacionado dentro de la misma sesión, activa la ejecución de un bloque de acción de recuperación para la aplicación en la que se ha tenido éxito. 2) Los mensajes entrantes se consideran procesados correctamente por la solución en los siguientes casos: cuando todas las aplicaciones de destino (CS, PS y NS) reciben el mensaje que está correlacionado dentro de la misma sesión o cuando al menos CS y PS lo reciben.

Los fallos en Port 5 no invalidan la ejecución de la solución, solamente activan un bloque de acción de recuperación cuyo efecto

es crear un registro que indica que el cliente no pudo ser notificado. El diseño de los bloques de acción de recuperación no es uno de los objetivos de este artículo. 3) La tercera restricción que asumimos en esta solución EAI es que la información de las facturas y de los pedidos debe llegar a las aplicaciones de destino a mas tardar en 24 horas.

Con el objetivo de cumplir con estas restricciones, hemos añadido a la solución EAI dos MEPs y tres reglas, cf. Figura 5. El primer MEP define dos puertos de entrada y dos de salida. Esto implica que una instancia del MEP se considera completa cuando se encuentra en el log el mensaje que está correlacionado dentro de la misma sesión para todos estos puertos. En segundo MEP también involucra el Port 5 en la lista de puertos de salida; esto da lugar a otra alternativa para que la solución EAI se complete con éxito. Siempre que el Exchange Engine detecta un MEP incompleto, el Rule Engine evalúa las reglas y activa los bloques de acciones de compensación correspondientes.

6. Conclusiones

En este artículo hemos presentado una propuesta de arquitectura para soluciones EAI basadas en sistemas de soporte a procesos, con tolerancia a fallos. Comenzamos explicando que las soluciones que otros autores ofrecen se centran en el modelo de ejecución basado en procesos y, consecuentemente, suelen consumir recursos inútilmente. Argumentamos también que desde esta perspectiva, el modelo de ejecución basado en tareas es una alternativa más eficiente y lo hemos explorado. Hemos presentado nuestra propuesta de arquitectura para tolerancia a fallos por medio de su metamodelo, que incluye un monitor para detectar los fallos y activar bloques de acción de recuperación. Hemos presentado una semántica de fallos que incluye tres de los fallos más importantes en las soluciones EAI y los hemos tratado. Hemos explicado también, aunque brevemente, como configurar los Message Exchange Patterns y reglas por medio de un lenguaje basado en Evento-Condición-Acción. Los MEPs incompletos provocan la activación de reglas para ejecutar acciones de recuperación. Para explorar la viabilidad de nuestras ideas hemos usado un ejemplo (caso de estudio) de solución EAI con soporte a tolerancia a fallos, para una empresa ficticia.

Referencias

[1] G. Alonso, C. Hagen, D. Divyakant, A. El Abbadi, and C. Mohan. Enhancing the fault tolerance of workflow management systems. *IEEE Concurrency*, 8(3):74–81, 2000.

[2] A. Avizienis, J.-C. Laprie, and B. Randell. Fundamental concepts of dependability. Technical Report CS-TR–739, Newcastle University, UK, 2001.

[3] Algirdas Avižienis. *IEEE Transactions on Software Engineering*, SE-I 1(12):1491–1501, December 1985.

[4] R.H. Campbell and B. Randell. Error recovery in asynchronous systems. *IEEE Trans. Soft. Eng.*, 12(8):811–826, 1986.

[5] M.Y. Chen, A. Accardi, E. Kiciman, J. Lloyd, D. Patterson, A. Fox, and E. Brewer. Path-based faliure and evolution management. In *Proc. Int'l Symp. Netw. Syst. Des. and Impl.*, page 23, 2004.

[6] D. Chiu, Q. Li, and K. Karlapalem. A meta modelng approach to workflow management systems supporting exception handling. *Inf. Syst.*, 24(2):159–184, 1999.

[7] G. Dobson. Using ws-bpel to implement software fault tolerance for web services. In *EUROMICRO '06: Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 126–133. IEEE Computer Society, 2006.

[8] G. Dunphy and A. Metwally. *Pro BizTalk 2006*. Apress, 2006.

[9] V. Ermagan, I. Kruger, and M. Menarini. A fault tolerance approach for enterprise applications. In *Proc. IEEE Int'l Conf. Serv. Comput.*, volume 2, pages 63–72, 2008.

[10] Apache Foundation. Apache Camel: Book In One Page, 2008.

[11] C.N. Hadjicostis and Verghese G.C. Monitoring discrete event systems using petri net embeddings. In *Proc. 20th Int'l Conf. Appl. and Theory of Petri Nets*, pages 188–207, 1999.

[12] C. Hagen and G. Alonso. Exception handling in workflow management systems. *IEEE Trans. Softw. Eng.*, 26(10):943–958, 2000.

[13] G. Hohpe and B. Woolf. *Enterprise Integration Patterns - Designing, Building, and Deploying Messaging Solutions.* Addison-Wesley, 2003.

[14] J.-C. Laprie. Dependability - its attributes, impairments and means. In *Predicting Dependable Computing Systems*, pages 3–24. 1995.

[15] L. Li, C.N. Hadjicostis, and R.S. Sreenivas. Designs of bisimilar petri net controllers with fault tolerance capabilities. *IEEE Trans. Syst. Man Cybern. Part A: Syst. Humans,*, 38(1):207–217, 2008.

[16] A. Liu, L. Huang, Q. Li, and M. Xiao. Fault-tolerant orchestration of transactional web services. In *Proc. Int'l Conf. Web Inf. Syst. Eng.*, pages 90–101, 2006.

[17] A. Liu, Q. Li, L. Huang, and M. Xiao. A declarative approach to enhancing the reliability of bpel processes. In *Proc. IEEE Int'l Conf. Web Services*, pages 272–279, 2007.

[18] C. Liu, M.E. Orlowska, X. Lin, and X. Zhou. Improving backward recovery in workflow systems. In *Proc. 7th Int'l Conf. Database Syst. Adv. Appl.*, page 276, 2001.

[19] Duncan Mackenzie. Building distributed applications: Architectural options for asynchronous workflow , revised 03/jun/2010, 2001.

[20] D. Messerschmitt and C. Szyperski. *Software Ecosystem: Understanding an Indispensable Technology and Industry.* MIT Press, 2003.

[21] H.G. Molina and K. Salem. Sagas. *SIGMOD Rec.*, 16(3):249–259, 1987.

[22] MuleSource. Mule 2.x User Guide, 2008.

[23] OASIS. Web Services Business Process Execution Language Version 2.0 Specification, 2007.

[24] C. Peltz. Web services orchestration: a review of emerging technologies, tools, and standards. Technical report, Hewlett-Packard Company, 2003.

[25] TIBCO. Tibco application integration software, Jun 2009.

[26] M. Wright and A. Reynolds. *Oracle SOA Suite Developer's Guide.* Packt Publishing, 2009.