Un Marco de Referencia para Comparar ESBs desde la Perspectiva de la Integración de Aplicaciones *

Rafael Corchuelo[1], Rafael Z. Frantz[2], Jesús González[3]

[1] Universidad de Sevilla, ETSI Informática
Avda. Reina Mercedes, s/n. Sevilla 41012
corchu@us.es
[2] Universidade Regional do Noroeste do Estado do Rio Grande do Sul (Unijuí)
São Francisco, 501. Ijuí 98700-000 RS (Brasil)
rzfrantz@unijui.edu.br
[3] Intelligent Dialogue Systems, S.L. (Indisys)
Edificio CREA, Avda. José Galán Merino, s/n. Sevilla 41015
j.gonzalez@indisys.es

Resumen. Los ecosistemas software actuales combinan aplicaciones desarrolladas sobre las más dispares plataformas. Con frecuencia aparece la necesidad de integrar algunas de esas aplicaciones para que puedan trabajar de forma conjunta y producir valor añadido, lo que supone retos tanto desde el punto de vista conceptual como técnico. Los ESBs son herramientas muy utilizadas en este contexto puesto que ofrecen una respuesta reutilizable a gran parte de estos retos. Por desgracia, no conocemos ningún marco de referencia que permita a los ingenieros del software comparar estas herramientas en este contexto particular. Si a esto le añadimos la disparidad de conceptos y vocabularios usados por cada una de ellas, el resultado es que la inversión en tiempo requerida para tomar una decisión puede ser bastante alta. En este artículo ofrecemos un resumen amplio del marco de referencia para comparar ESBs que hemos diseñado sobre la base nuestra experiencia con dos proyectos de integración en ecosistemas software reales.

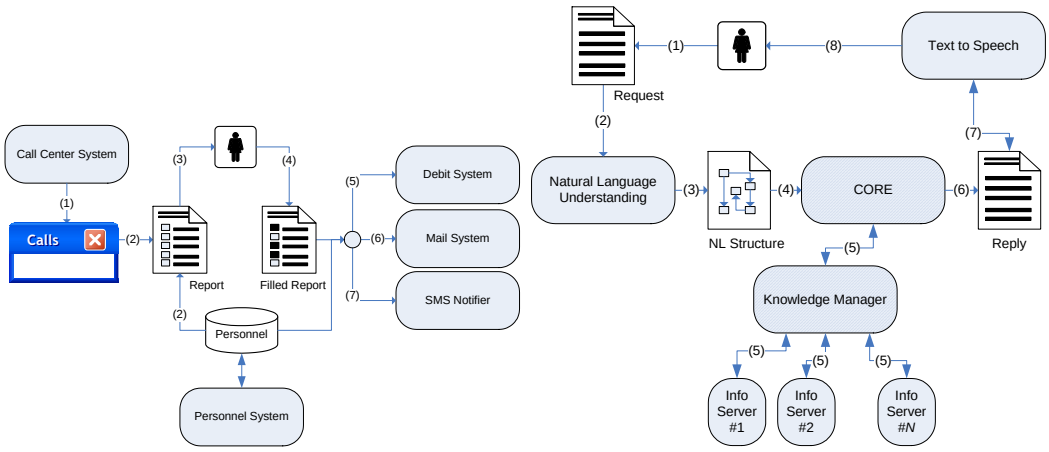
Palabras clave: integración de aplicaciones, ESBs, marco de comparación.

1. Introducción

En las empresas actuales es muy habitual que convivan aplicaciones que han sido adquiridas o desarrolladas conforme dichas empresas han evolucionado y han ido descubriendo nuevos requisitos, dando lugar a ecosistemas software que no siempre son fáciles de gestionar [11]. Un problema frecuente en estos ecosistemas es integrar dos o más aplicaciones de forma que los datos que manejan por separado estén sincronizados o que puedan colaborar para ofrecer nueva funcionalidad o nuevas vistas de datos [8]. Según un reciente informe de IBM los gastos de integración superan en una proporción de entre cinco y veinte a los de desarrollo de nueva funcionalidad [17]. No es de extrañar, por lo tanto, la enorme popularidad que las herramientas para construir buses de servicios empresariales (ESBs) están ganando en este contexto, ya que ofrecen la infraestructura necesaria para integrar los sistemas más dispares [3].

En nuestra revisión de la tecnología, hemos estudiado las siguientes herramientas: Camel [5], Mule [4], ServiceMix [2], Spring Integration [14] y BizTalk [19]. Las hemos elegido puesto que actualmente son de las más populares y, además, en el caso de ServiceMix, al estar basada en JBI [2] podemos considerarla como el representante canónico de todos los ESBs que también lo implementan, e.g., OpenESB, Fuse ESB o JBoss ESB. En mayor o menor grado, todas permiten implementar soluciones de integración basadas en el patrón arquitectónico Pipes&Filters [6]. Siguiendo este

* Financiado parcialmente por el Plan Nacional de I+D+I (expediente TIN2007-64119) y la Órden de Incentivos de la Junta de Andalucía (expediente P07-TIC-02602). Parte de esta financiación procede de fondos FEDER. El trabajo de R.Z. Frantz ha sido financiado por la Evangelischer Entwicklungsdienst (EED).



(a) Sistema de gestión de llamadas. (b) Sistema de orientación laboral.

Fig. 1. Dos ecosistemas reales.

patrón, una solución de integración se puede ver como el diseño de un conjunto de mensajes que fluyen a través de tuberías entre diversos filtros. Qué es un mensaje, un filtro o una tubería depende por completo de la infraestructura elegida y del problema a resolver. Por ejemplo, un mensaje puede ser desde un documento XML hasta un objeto Java serializado; un filtro puede ser desde objeto COM que realiza una transformación de mensajes sencilla hasta un servicio web que interactua con una aplicación de negocio; una tubería puede ser desde un sistema de archivos al que se accede por FTP hasta un canal de comunicación SOAP/HTTP. Esta gran flexibilidad es la que hace que las herramientas para construir ESBs sean la opción habitual para implementar soluciones de integración en ecosistemas software muy heterogéneos.

Recientemente, hemos trabajado con estas herramientas para diseñar dos soluciones de integración en un par de ecosistemas software reales, a saber: el sistema de gestión de llamadas telefónicas de Unijuí y un sistema de orientación laboral para una institución pública. Esta experiencia nos ha permitido profundizar en estas herramientas y diseñar un marco de referencia que permite compararlas usando propiedades que se pueden evaluar de forma objetiva. Hasta donde alcanza nuestro conocimiento, no existe ninguna propuesta similar, lo que dificulta en gran medida el trabajo de los ingenieros del software a la hora de tomar una decisión con respecto a qué herramienta utilizar en función a las características del problema de integración al que se enfrentan.

El resto del artículo está organizado de la siguiente forma: en la sección 2, presentamos dos ecosistemas reales que nos han ayudado a diseñar el marco de referencia que proponemos; en las secciones 3, 4 y 5 es donde describimos las propiedades de nuestro marco en relación con el alcance de las soluciones de integración, las capacidades de modelado y diversos aspectos de carácter puramente técnico; finalmente, mostramos nuestras conclusiones más importantes en la sección 6.

2. Motivación

Nuestro interés por desarrollar el marco de referencia que presentamos en este artículo surge a partir del estudio llevado a cabo para diseñar dos soluciones de integración en el contexto de dos ecosistemas reales y complejos completamente diferentes. Estas experiencias nos han servido para identificar cuáles son las propiedades a tener en cuenta a la hora de tomar una decisión sobre cuál es el ESBs más adecuado y esperamos que nos ayude en proyectos futuros para facilitar este trabajo.

El primer ecosistema software explorado fue el de gestión de llamadas telefónicas de la Universidad Unijuí, cf. Fig. 1. Existe un sistema central de gestión de llamadas denominado Call Center

System que almacena en una base de datos información sobre todas las llamadas realizadas por los empleados de la universidad desde los teléfonos que ésta pone a su disposición (1). Cada mes se realiza un análisis de esta base de datos con el objetivo de encontrar aquellas llamadas que suponen un coste para la universidad y que no queda claro si se trata de llamadas de trabajo o personales (2). El resultado de este análisis es un informe que se remite a los empleados para que marquen en él las llamadas personales (3). Los empleados deben devolver el informe relleno a los servicios de gestión (4), que procederán a realizar los cargos oportunos a través de una aplicación denominada Debit System (5) y a notificar el hecho por correo electrónico y SMS (6, 7). Además, también existe un sistema denominado Personnel System a partir del que se obtienen los números de teléfono, direcciones de correo y otros datos personales de los empleados de la universidad. Nuestro objetivo en este caso ha sido diseñar una solución de integración que permita automatizar el proceso con el objeto de aumentar su eficiencia y reducir los errores que se producen al manejar toda esta información con procedimientos artesanales.

Nuestro segundo caso de estudio está relacionado con el desarrollo de un asistente virtual interactivo para la orientación laboral que la empresa Indisys está realizando en colaboración con otras para una institución pública, cf. Fig. 1. El asistente virtual es una aplicación web que responde a preguntas frecuentes del estilo "¿dónde encuentro información sobre ofertas de trabajo?" o "me he quedado en paro; ¿qué papeles tengo que rellenar?". Este ecosistema cuenta con un sistema llamado Natural Language Understanding que es capaz de reconocer frases como las anteriores y convertirlas en una estructura de datos con semántica (1, 2, 3). CORE es el sistema que Indisys está implementando y se encarga analizar la estructura anterior con el objetivo de determinar si es necesario continuar el diálogo para que el usuario proporcione más información, e.g., "¿cuál es su sector profesional?" o "¿en qué provincia reside?", o por el contrario se tienen todos los datos para interrogar los servidores de información de la institución pública (5). En este último caso, el problema es que estos servidores son tremendamente heterogéneos, por lo que otra empresa está desarrollando un sistema para integrarlos y proporcionar una fachada de acceso denominada Knowledge Manager. Sea cual sea el caso, CORE produce una respuesta textual (6) que el sistema Text to Speech se encarga de transformar en voz (7, 8).

Destacamos tres aspectos fundamentales de estos proyectos:

Aplicaciones: La característica común de todas las aplicaciones integradas en nuestro primer caso de estudio es que constituyen sistemas independientes que fueron desarrollados sin pensar nunca en la posibilidad de que tuvieran que ser integrados con otros. Esto implica que no proporcionan interfaces de programación, por lo que en el caso del Personnel System es necesario acceder directamente a su base de datos, mientras que en el caso del Call Center System, que es un sistema propietario, es necesario interactuar a través de su interfaz de usuario usando un sistema de wrapping [1]. En el segundo caso de estudio, por el contrario, casi todos los sistemas proporcionan una interfaz de programación que facilita la integración, salvo en el caso de los sistemas de información de la institución pública, que son completamente dispares.

Brechas: Las principales brechas a salvar están relacionadas con la tecnología, los modelos de datos y la representación de los mismos. En los dos casos de estudio, hemos encontrado una amplia variedad de tecnologías, desde bases de datos a las que se puede acceder mediante JDBC hasta interfaces de programación que utilizan protocolos propietarios. No es de extrañar, pues, que los modelos de datos usados sean completamente dispares y sean necesarias con frecuencia transformaciones tanto en el esquema como en la representación de los datos.

Restricciones: En todos los casos, las aplicaciones integradas no han sufrido ningún cambio. Esta restricción ha sido motivada en unos casos por el hecho de tratarse de aplicaciones propietarias como el Call Center System; en otros porque el coste de la reingeniería necesario para que ofreciesen una interfaz de programación más adecuada era inasumible. Otra restricción importante ha sido mantener las aplicaciones integradas completamente desincronizadas con el objetivo de que puedan continuar siendo operadas y administradas de forma completamente independiente.

Propiedad	Camel	Mule	ServiceMix	Spring Int.	BizTalk
Contexto	EAI	EAI	EAI	EAI	EAI/B2BI
Patrón arquitectónico	Filters	Pipes/Filters*	Pipes/Filters*	Filters	Pipes/Filters
Nivel	PSM	PSM	PSM	PSM	PSM
Transacciones	ST-Filtro	ST-Filtro	ST-Filtro/ST-Solución	–	ST-Filtro/LT-Filtro
RoutingSlip dinámico	No	No	No	No	No
Extensiones	No	No	No	No	Sí
Adaptadores de aplicación	No	No	No	No	Sí
Tipo de modelo	O/D-IoC	O/D-IoC	O/D-IoC	O/D-IoC	D-Gráf. y XML
Licencia	L-A	L-BL	L-A	L-A	L-P

* Tan sólo ofrecen soporte parcial para el diseño de filtros.

Tabla 1. Propiedades relacionadas con el alcance de las herramientas.

3. Alcance de las Herramientas

En esta sección y las dos siguientes, mostramos gran parte de las propiedades que forman nuestro marco de referencia. La primera categoría está relacionada con el alcance de las herramientas puesto que se trata de propiedades cuya ausencia puede dificultar enormemente e incluso invalidar una propuesta ya que para suplirlas es necesario implementar extensiones cuyo coste de desarrollo creemos que puede ser inabordable en la mayoría de los casos. Entre las más importantes, destacamos las siguientes, cf. tabla 1:

Contexto: Suele ser habitual distinguir entre los siguientes contextos de integración: Enterprise Application Integration (EAI), en donde el énfasis es integrar aplicaciones con el objetivo de sincronizar sus datos o de implementar nuevas funcionalidades; Enterprise Information Integration (EII), cuyo énfasis están en proporcionar una vista en vivo de los datos que manejan las aplicaciones integradas; Extract, Transform, and Load (ETL), que busca proporcionar vistas materializadas de dichos datos sobre la que aplicar técnicas extracción de conocimiento [18]. En todos los casos anteriores, se asume implícitamente que las aplicaciones integradas forman parte de una misma organización. Recientemente cada vez se le está prestando más atención al problema de integrar aplicaciones pertenecientes a distintas organizaciones con el objeto de implementar procesos de negocio inter-organizacionales; a este contexto se suele hacer referencia como Business to Business Integration (B2BI). También recientemente, han cobrado importancia los denominados mashups, que son aplicaciones que se ejecutan en un navegador web e integran datos proporcionados por diversas aplicaciones web. De nuestro análisis se desprende que casi todas las soluciones estudiadas se encuentran en el contexto de EAI, con la única excepción de BizTalk, que también tiene en cuenta el contexto B2BI.

Patrón arquitectónico: Como ya sabemos, Pipes&Filters es el patrón por excelencia en nuestro campo de interés. Por lo tanto, parece razonable esperar que las herramientas para la construcción de ESBs proporcionen lenguajes específicos de dominio para diseñar tanto tuberías como filtros. Por desgracia, no es así ya que Camel y Spring Integration no proporcionan soporte alguno para el diseño de tuberías y Mule y ServiceMix tan sólo proporcionan un soporte parcial para el diseño de filtros. En el caso de Mule, este soporte se reduce a unas cuantas tareas de transformación o enrutado de mensajes que se deben combinar siempre de forma lineal; en el caso de ServiceMix, la herramienta, como tal, no ofrece ninguna ayuda para construir los filtros, pero existe un componente JBI que proporciona la implementación de algunas tareas comunes. En los dos casos, es muy habitual que el núcleo de los filtros se diseñe directamente en Java debido a las limitaciones de las herramientas.

Nivel de abstracción: Trabajar con modelos independientes de la plataforma (PIM) permite diseñar soluciones estables tan independientes como resulta posible de la tecnología utilizada para implementarlas y su inevitable evolución. Al trabajar con modelos PIM es necesario contar, además, con herramientas capaces de transformarlos en modelos dependientes de la plataforma sobre la que se quiere realizar la implementación (PSM) [10]. Por desgracia, ninguna de las herramientas estudiadas permite realizar una separación clara entre los niveles PIM y PSM dado que todas obligan al diseñador a trabajar directamente sobre las plataformas Java o .NET.

Transacciones: Las transacciones permiten diseñar soluciones robustas capaces de hacer frente a fallos que puedan ocurrir durante la ejecución de una solución de integración. Es habitual distinguir entre transacciones a corto plazo (ST) o a largo plazo (LT): las primeras suelen implementarse usando el conocido protocolo Two-Phase Commit que, básicamente, consiste en llevar a cabo aquellas acciones que requieren cambios de estado tan sólo cuando todas las partes implicadas han confirmado que pueden llevarlas a cabo sin problema [13]; las segundas, por el contrario, ejecutan las acciones necesarias conforme sea posible y, en caso de que alguna falle, ejecutan a continuación acciones de compensación cuyo objetivo es deshacer los efectos de las primeras o, en caso de que esto no sea posible, paliarlos [7,12]. En el contexto de la integración de aplicaciones, también resulta útil clasificar las transacciones en función a su alcance, según lo cuál se distingue entre transacciones de filtro o de solución: las primeras son aquéllas que garantizan que un filtro logra ejecutar su tarea o de lo contrario cualquier cambio que haya podido realizar hasta el momento de detectar un fallo queda invalidado; las segundas son aquéllas que garantizan esta propiedad desde el punto de una solución completa. Casi todas las propuestas soportan transacciones a corto plazo, con la excepción de BizTalk que también soporta transacciones a largo plazo y de Spring Integration que no proporciona ningún soporte propio; con la excepción de Camel, todas soportan transacciones tanto a nivel de filtro como de solución, pero, por desgracia, ninguna soporta transacciones a largo plazo a nivel de solución.

RoutingSlip dinámico: El patrón RoutingSlip permite enrutar un mensaje a través de una secuencia de filtros que no son conocidos a priori [8]. Para conseguirlo se puede hacer que los mensajes incluyan en su cabecera la lista de filtros o que esta información esté en algún fichero de configuración. Gracias a esto es posible definir una ruta para los mensajes después de haber diseñado la solución de integración. La posibilidad de cambiar dinámicamente dicha lista durante la ejecución de un filtro en la solución, para, por ejemplo, en función de un determinado resultado añadir o quitar un filtro, hace que este patrón sea de gran utilidad práctica. Por desgracia, entre las herramientas estudiadas apenas Camel a partir de la versión 1.3 y ServiceMix soportan el patrón RoutingSlip, pero todavía, ninguno de ellos permiten cambiar de forma dinámica la lista de filtros, que debe ser asignada en el mismo momento en que se crea cada mensaje y se mantiene intacta durante todo el procesamiento que éste deba sufrir.

Extensiones: Las extensiones proporcionadas por las herramientas aportan tareas específicas para un determinado dominio de negocio. Suelen dar soporte a estándares definidos para dichos dominios y facilitan mucho el diseño de una solución de integración dentro de los mismos. El hecho de no haber disponible una extensión para un dominio puede resultar demasiado costoso para una empresa a la hora de crear una solución de integración para su problema. Por desgracia, sólo BizTalk ofrece extensiones para el área de la salud, el comercio electrónico, las finanzas y las cadenas de aprovisionamiento.

Adaptadores de aplicación: Los adaptadores de aplicación permiten conectar sistemas grandes enteros a una solución de integración. Esto permite una comunicación de más alto nivel entre la solución y la aplicación a fin de intercambiar información. Un ejemplo claro es el adaptador de aplicación ofrecido por la empresa iWay [9] para usar el ERP Peoplesoft desde dentro de BizTalk. Creemos que el coste de desarrollo de estos tipos de adaptadores, para una empresa que va a diseñar una solución de integración, puede resultar prohibitivo, además de que, en muchos casos, no se tiene el conocimiento necesario sobre la tecnología para la que hay que diseñar el

Propiedad	**Camel**	**Mule**	**ServiceMix**	**Spring Int.**	**BizTalk**
Vistas	–	–	–	Wrapper/Proceso/Solución	–
Card. filtros	$1:N$	$1:N$	$1:N$	$1:N$	$N:M$
Card. tareas	$1:N$	$1:1$‡	–*	$1:N$	$1:1$
Card. localidades	$N:M$-Comp	$N:M$-Comp	$N:M$-Comp	$N:M$-Comp	$1:1$
Correlación	No	No	No	No	Sí
Tipos de puertos	PD	PD	PD/PM†	PD	PD/PM†
Adaptadores en puertos	1:1	1:1	1:1	1:1	N:1
Puertos bidireccionales	No	No	Sí	No	Sí
Pipeline en puertos	No	Sí†	No	Sí†	Sí
Filtros con estado	No	No	No	No	No
Tareas con estado	Sí	Sí	Sí	Sí	No
Diseño de tareas	No	No	No	No	Sí

* Las tareas son proporcionadas por un componente externo.
† Con limitaciones importantes.
‡ Salvo en el flujo de salida, en el que pueden haber tareas con $1:N$.
Tabla 2. Propiedades relacionadas con las capacidades de modelado.

adaptador. Por desgracia, entre las herramientas estudiadas, sólo hemos encontrado adaptadores de aplicación para BizTalk.

Tipo de modelo: De acuerdo con el tipo de modelo de la herramienta, se permite diseñar las soluciones de integración de forma operativa (O) o declarativa (D). Cuando el diseño es operativo, la herramienta proporciona una biblioteca de clases que los programadores pueden combinar para crear sus soluciones de integración; por el contrario, cuando es declarativo el programador puede trabajar a un nivel de abstracción mucho más alto. La primera forma es usando un lenguaje específico de dominio con representación en XML o gráfica que después será traducido automáticamente a código ejecutable; la segunda es por medio de ficheros de configuración XML que después serán interpretados por un motor de inversión de control (IoC), e.g., Spring [16]. Desde este punto de vista, Camel, Mule, ServiceMix y Spring Integration soportan tanto modelos operativos como declarativos mediante inversión de control; BizTalk es la única herramienta que ofrece un modelo declarativo tanto basado en XML como gráfico.

Licencia: El tipo de licencia de una herramienta puede invalidar su adopción en una empresa por cuestiones políticas o económicas. En la definición de este marco de referencia hemos clasificado las licencias en: licencias Apache (L-A), otras licencias basadas en licencias libres (L-BL) y licencias propietarias (L-P).

4. Capacidades de Modelado

En esta sección proporcionamos detalles sobre un conjunto de propiedades que no son tan críticas como las anteriores. La razón es que en caso de carecer de ellas aún es posible diseñar una solución de integración efectiva con un coste razonable, pero quizás el diseño sea mucho más complejo y menos intuitivo, lo que puede tener, evidentemente, un efecto negativo sobre el mantenimiento posterior. Entre las más importantes, destacamos las siguientes, cf. tabla 2:

Vistas: En una solución de integración, es posible distinguir las siguientes vistas: wrapper, que es la vista que muestra aquellos filtros cuyo objetivo es comunicar una aplicación con una solución

de integración o al revés; proceso, que incluye aquellos filtros que implementan una tarea propia de la solución de integración; puertos, que se centra en el diseño de las tuberías de comunicación y los adaptadores que los filtros requieren para acceder a las mismas; solución, que es la más general puesto que describe los flujos necesarios entre un conjunto de filtros para integrar varias aplicaciones. Por desgracia, ni Camel, ni Mule, ni ServiceMix, ni BizTalk ofrecen la posibilidad de modelar ninguna de estas vistas, por lo que un filtro puede acabar implementando un wrapper, un proceso o una solución completa, sin que sea posible documentar esto de forma explícita en los modelos resultantes. Spring Integration es la más completa de la estudiadas ya que distingue entre las vistas de wrapper, proceso y solución.

Cardinalidad de los filtros: Algunos filtros pueden requerir mensajes de distintas fuentes para poder desempeñar su labor. Un situación común se presenta cuando es preciso tratar las distintas partes de un mensaje utilizando filtros diferentes y posteriormente combinar los resultados. Por desgracia, tan sólo BizTalk permite diseñar filtros capaces de tomar información de varias fuentes. Por el contrario, todas las propuestas analizadas permiten que un filtro envíe información a varias tuberías de salida simultáneamente. En este punto es interesante destacar que gracias a herramientas como los motores de BPEL es posible diseñar filtros capaces de tomar información de varias fuentes de una forma razonablemente simple. Usando una herramienta de este tipo sería, por lo tanto, posible, suplir las carencias de las herramientas analizadas, aunque a costa de introducir un nuevo lenguaje en el proceso de modelado.

Cardinalidad de tareas: Los filtros están formados por tareas más simples que permiten construir mensajes, transformarlos, enrutarlos o interactuar con las tuberías. En nuestra experiencia, es habitual que algunas tareas necesiten información contextual proveniente de alguna aplicación para poder tratar los mensajes que le llegan; un ejemplo claro es cuando los mensajes procesados incluyen algún identificador y es preciso consultar a una aplicación qué datos corresponden al mismo. Por desgracia, ninguna de las propuestas examinadas permite tareas con varias entradas, mientras que Camel y Spring Integration sí que permiten tareas con varias salidas. El caso de ServiceMix es un poco especial pues, como comentamos antes, las tareas son proporcionadas por un componente JBI externo; en cualquier caso, este componente tan sólo permite tareas con una entrada aunque pueden tener varias salidas.

Cardinalidad de localidades: El término localidad hace referencia a la ubicación física sobre la que se implementa una tubería, e.g., una carpeta en un sistema de archivos o un servidor de correo electrónico. Con la excepción de BizTalk, todas las soluciones estudiadas permiten que varios filtros lean o escriban mensajes desde/en una localidad compartida. Generalmente es interesante distinguir entre dos tipos de lectura: con competencia (Comp), en cuyo caso tan sólo uno de los filtros puede leer en cada momento de una determinada localidad, o con replicación (Repl), en cuyo caso todos los filtros pueden leer al mismo tiempo. Todas las soluciones implementadas permiten lectura con competencia, pero ninguna la lectura con replicación.

Correlación de mensajes: Dado que no podemos asumir sincronía alguna entre las aplicaciones integradas, es muy común que los mensajes lleguen a los filtros de forma desordenada, por lo que es responsabilidad de los mismos correlacionarlos de forma que aquellos mensajes que son complementarios sean tratados siempre de forma conjunta. Esta necesidad es mucho más imperiosa en aquellos casos en que es posible diseñar filtros o tareas con múltiples entradas, por lo que no es de extrañar que tan sólo BizTalk soporte directamente la correlación de mensajes.

Tipos de puertos: Una solución de integración puede tener dos tipos de puertos, a saber: puertos de datos (PD) y puertos de mensajes (PM). Los puertos primeros se encargan de conectar una aplicación con un filtro a través de una localidad; los datos que fluyen por estos puertos pertenecen a la aplicación que los produce o los consume. Los puertos de mensaje, por el contrario, son aquellos que forman parte exclusiva de las soluciones de integración y los datos que fluyen por ellos tienen un formato de mensaje bien definido, es decir, incluyen cabeceras y posibles adjuntos además de los datos en sí. Un ejemplo claro de información de cabecera es el tipo del mensaje, la dirección de retorno, la lista de filtros por los que debe pasar en caso de soportar el patrón RoutingSlip [8], el identificador de secuencia, etcétera. Al permitir que dos o más fil-

tros se puedan comunicar mediante puertos de mensajes se consigue que toda esta información extra fluya de forma natural entre los mismos, lo que, entre otras cosas, facilita la trazabilidad. Por desgracia, sólo ServiceMix y BizTalk ofrecen puertos de mensaje, aunque con limitaciones dado que en el primer caso sólo pueden conectar a filtros que se encuentran dentro de la misma máquina y en el segundo caso sólo permiten comunicación síncrona entre los mismos.

Adaptadores en puertos: La posibilidad de que un puerto tenga múltiples adaptadores le permite recibir/enviar información desde/a dos o más localidades. La ligadura de un puerto con múltiples localidades ayuda a mantener el modelo sencillo e intuitivo, puesto que, en caso que sólo se pueda ligar un puerto a una localidad, el modelado de un filtro puede resultar más complejo. Un ejemplo claro es cuando el filtro puede recibir información desde dos fuentes. En estos casos si el puerto no permite múltiples adaptadores, habrá que modelar dos puertos, uno para cada fuente. Además, la estructura interna del filtro también tendrá que ser modificada de forma que haya una tarea al principio sólo para recibir los mensajes de los puertos y darle paso a la próxima tarea. Por desgracia, sólo BizTalk permite tener puertos con múltiples adaptadores y con algunas limitaciones ya que esto sólo es posible en el caso de los puertos de entrada.

Puertos bidireccionales: Los puertos bidireccionales permiten un modelado más sencillo y directo, especialmente, en los casos que un filtro envía un mensaje a otro y espera una respuesta. Dichos puertos permiten que la petición y la respuesta sean tratadas en el mismo puerto, el que también permite, hacer la correlación de los mensajes en el propio filtro que las ha procesado de forma más sencilla. El hecho de no haber puertos bidireccionales obliga a usar dos puertos diferentes, uno de entrada y otro de salida, lo que también dificulta la tarea de correlación del mensaje original con la respuesta. Por desgracia, de las herramientas estudiadas sólo ServiceMix y BizTalk permiten puertos bidireccionales.

Pipeline en puertos: Un pipeline permite a un puerto ejecutar un pre/pos procesamiento en serie sobre el mensaje antes y/o después de que las tareas internas a un filtro lo procesen. El pipeline de los puertos de entrada suele preparar el mensaje para que las tareas internas al filtro puedan procesarlo, mientras que el pipeline de los puertos de salida suele dejar el mensaje en un formato que se pueda transmitir al próximo filtro. El hecho de no soportar pipelines en los puertos obliga a añadir tareas al principio y/o al final del filtro para hacer dichos pre/posprocesamientos. Ejemplos claros de preprocesamientos en un pipeline de un puerto de entrada son la decodificación o descompresión de los mensajes recibidos, la validación de los mismos, la autentificación del remitente, etcétera. Un procesamiento común y exclusivo a los puertos de datos, tanto en un pipeline de los puertos de entrada como en un de los de salida, es el mapeo de datos brutos a mensajes y viceversa. Entre las herramientas estudiadas, Mule, Spring Integration y BizTalk son las que ofrecen algún soporte al concepto de pipeline, siendo BizTalk la que incluye un modelo de pipeline más elaborado y más componentes para construirlos.

Filtros con estado: Un filtro puede necesitar guardar información útil en próximas ejecuciones del mismo. Esto permite, por ejemplo, guardar una lista de los últimos mensajes recibidos que evite procesar en el futuro mensajes que son semánticamente equivalentes; en otras ocasiones puede servir para guardar resultados que son costosos de calcular, evitando así que al recibir un mensaje parecido los cálculos se tengan que repetir. Por lo tanto, es deseable que los filtros puedan guardar estado, pero por desgracia ninguna de las herramientas estudiadas lo soporta directamente.

Tareas con estado: De la misma forma que para los filtros, permitir que todas las tareas puedan guardar su estado, puede resultar interesante. Una tarea puede guardar información de estado persistiendo su contexto de ejecución o guardando un mensaje con información contextual. Un ejemplo común de tarea que puede guardar estado es el agregador y el resecuenciador [8]. El agregador se encarga de volver a unir dos o más mensajes, mientras que el resecuenciador recibe mensajes desordenados y los ordenada. Ambos tipos de tareas tienen que guardar los mensajes que reciben hasta que se pueda procesarlos. Entre las herramientas estudiadas tan sólo Camel, Mule, ServiceMix y Spring Integration ofrecen algunas tareas que pueden guardar estado, pero sólo en forma de mensajes.

Propiedad	Camel	Mule	ServiceMix	Spring Int.	BizTalk
Modelo ejecución	1 : 1	1 : 1	1 : 1	1 : 1	1 : 1
Mensajes tipados	No	No	No	No	Sí/No
Mensajes anómalos	Sí	No	No	Sí	Sí
Descargar ejecución	No	No	No	No	Sí
Patrón de comunicación	Sí*	No	Sí	No	No
Soporte para pruebas	Sí	No	No	No	Sí
Adjuntos	No	Sí	Sí	No	Sí
Diseño de adaptadores	No	No	No	No	Sí
Sistema de gestión	Sí	Sí	Sí	No	Sí
Soporte comercial	Sí	Sí	Sí	Sí	Sí

* No permite definir nuevos tipos de MEPs.

Tabla 3. Propiedades de carácter técnico.

Diseño de tareas: Cuando una herramienta no soporta un determinado tipo de tarea, generalmente hay que recurrir a una tarea general y configurarla mediante código ad-hoc. El problema es que estas tareas generales no hacen explícita la intención del diseñador, lo que puede resultar un problema añadido a la hora de entender los modelos. Desde este punto de vista es deseable que la herramienta ofrezca soporte al diseño de nuevas tareas reutilizables, haciendo explícita su intención. Por desgracia, de las herramientas estudiadas, sólo BizTalk proporciona una herramienta para diseñar nuevas tareas; en el resto de los casos es necesario recurrir a programación de bajo nivel.

5. Características Técnicas

En esta categoría hemos incluido aquellas propiedades que afectan a la facilidad de programación, al rendimiento o a la gestión de las soluciones de integración, por lo que su ausencia puede dificultar el despliegue y la operación de las mismas. Entre las más importantes, destacamos las siguientes, cf. tabla 3:

Modelo de ejecución: El modelo de ejecución de los filtros puede tener un impacto importante en el rendimiento de una solución de integración. El más sencillo consiste en asignar una hebra a cada mensaje o conjunto de mensajes que deben ser tratados de forma conjunta por un filtro; por supuesto, las hebras pueden tomarse de un pool para conseguir de esta forma mantener siempre bajo control la carga total de trabajo del servidor. Este es el modelo que implementan todas las herramientas estudiadas, pero presenta una deficiencia que creemos que puede afectar de forma negativa a la escalabilidad de las soluciones. El problema está relacionado con el hecho de que cuando una instancia de un filtro llega a un punto en el que no puede continuar ejecutando tareas, por ejemplo, porque es necesario esperar la llegada de un mensaje, la hebra queda ociosa durante un tiempo completamente indeterminado. De nuestra experiencia concluimos que un modelo capaz de ejecutar de forma asíncrona varias instancias de un mismo filtro sobre un pool de hebras sería mucho más efectivo. En la actualidad estamos trabajando en el diseño e implementación de este modelo con el objetivo de evaluar ambas alternativas y poder obtener conclusiones.

Mensajes tipados: En soluciones típicas de integración suele ser necesario realizar decenas de transformaciones a los mensajes; cualquier error en una de ellas puede dar lugar a un mensaje incoherente, por lo que es muy deseable que estos sean tipados. Por desgracia, salvo en el

caso de BizTalk, todas las herramientas examinadas asumen que los mensajes son objetos Java sobre los que no se realiza ninguna comprobación de tipo; en cualquier caso, también es posible tener mensajes sin tipo en BizTalk.

Mensajes anómalos: Cuando un mensaje presenta algún tipo de anomalía que hace imposible que sea procesado por un filtro, lo normal es que éste produzca una excepción y que el mensaje en cuestión se almacene de forma que pueda ser analizado por el administrador del sistema. Además, es muy deseable que estos mensajes también puedan ser tratados de forma automática de manera que se intente llevar a cabo algún tipo de acción correctiva en el mismo instante en el que se detectan. Por desgracia, ni Mule ni ServiceMix ofrecen ningún mecanismo que permita automatizar el tratamiento de estos mensajes.

Descargar la ejecución: El procesamiento de un mensaje por un filtro puede exigir aguardar la recepción de un segundo mensaje. Un situación común se presenta cuando es necesario enriquecer un mensaje con información que pertenece a una fuente externa. En estos casos la ejecución del filtro se queda parada durante un tiempo, completamente indeterminado, hasta que no llegue el otro mensaje. Por lo tanto, la posibilidad de descargar el contexto de ejecución del filtro permite ahorrar recursos que pueden ser fundamentales para la maquina. Por desgracia, de las herramientas estudiadas, tan sólo BizTalk permite descargar la ejecución.

Patrón de comunicación: El patrón para intercambio de mensajes (MEPs) permite definir tipos específicos de comunicación [15]. Una solución de integración puede utilizar distintos tipos de MEPs, como el unidireccional y sin respuesta (InOnly), el bidireccional con una respuesta obligatoria (InOut), el bidireccional con una respuesta opcional (InOptionalOut), etcétera. El uso de MEPs facilita la correlación entre los mensajes que llegan a un filtro y las respuestas obtenidas. Por este motivo, desde un punto de vista técnico, es deseable que la herramienta ofrezca soporte a dicho patrón y permita definir nuevos tipos de MEPs, además de los que ya pueda aportar. En nuestro estudio hemos visto que las únicas que soportan este patrón de comunicación son Camel y ServiceMix, aunque sólo ServiceMix permite definir nuevos tipos de MEPs.

Soporte para pruebas: Un soporte para pruebas permite diseñar aplicaciones sencillas y enfocadas para ayudar a validar el funcionamiento de una solución de integración. En nuestra experiencia hemos visto que diseñar pruebas desde cero puede resultar complejo para el programador, además de costoso para la empresa. Desde este punto de vista, es deseable que las herramientas para la integración ofrezcan algún soporte para pruebas puesto que ayudaría a reducir la complejidad y el coste. De las herramientas estudiadas, tan sólo Camel y BizTalk ofrecen soporte para diseñar pruebas.

Mensajes con adjuntos: Un mensaje puede llevar, además de la información de cabecera y cuerpo, otros objetos con información adjunta. La información adjunta no debe sufrir ningún procesamiento en la solución de integración, es decir, sólo representa información adicional que se transmite junto con el mensaje. Creemos que el hecho de separar la información adjunta del cuerpo del mensaje permite reducir su tiempo de procesamiento, puesto que la tarea al tener que acceder al cuerpo no tendrá que tratar los datos de los adjuntos. Además, en caso que los adjuntos estén separados del cuerpo, se puede utilizar el patrón ClaimCheck [8] para guardarlos en un medio persistente, mientras se procesa en mensaje, y después recuperarlos. De las herramientas estudiadas, las únicas que no soportan mensajes con adjunto son Camel y Spring Integration.

Diseño de adaptadores tecnológicos: Los adaptadores tecnológicos facilitan la comunicación a bajo nivel con una fuente de datos externa a la solución de integración, e.g, base de datos, fichero XML, canal JMS, etcétera. Todas las herramientas estudiadas aportan varios tipos de adaptadores tecnológicos para las tecnologías mas conocidas, como JDBC, SQLServer, JMS, POP3, FTP, etcétera. Para los casos que la herramienta no proporciona un determinado tipo de adaptador que se necesite, el programador puede desarrollarlo, aunque esto implique en tener un conocimiento más especializado sobre el tema. Por este motivo, desde un punto de vista técnico, es deseable que las herramientas aporten recursos que permitan crear dichos adaptadores

de una forma más sencilla e intuitiva. Por desgracia, sólo BizTalk proporciona una herramienta específica para hacer esto.

Sistema de gestión: Un sistema de gestión proporciona una vista de alto nivel de la solución de integración desplegada y que se está ejecutando. Además suele permitir monitorizar en tiempo real la ejecución de los filtros, las transacciones, el rendimiento, etcétera. Desde este punto de vista es deseable que las herramientas ofrezcan algún sistema que permita gestionar la solución de integración. La ausencia de dichos sistemas obliga a los programadores diseñar aplicaciones ad-hoc para realizar la gestión. Entre las herramientas que hemos estudiado, sólo Spring Integration no aporta ningún sistema de este tipo.

Soporte comercial: El soporte comercial ofrecido por las empresas creadoras de las herramientas propietarias, o por aquéllas que están por detrás de los proyectos de herramientas de código abierto, puede resultar de gran interés a quién va a utilizar la tecnología. La totalidad de las herramientas estudiadas ofrecen soporte comercial.

6. Conclusiones

La principal conclusión del estudio que hemos llevado a cabo es que ninguna de las herramientas analizadas es ideal, en el sentido de que ninguna de ellas tiene valores óptimos para todas las propiedades examinadas. También hemos detectado algunas carencias importantes en todas ellas. Quizá una de las principales es que ninguna permite desarrollar modelos independientes de la plataforma, lo que liga las soluciones a la tecnología disponible en cada momento y puede dificultar su mantenimiento cuando la tecnología evolucione. También destaca el hecho de que ninguna de las propuestas estudiadas soporte transacciones a largo plazo a nivel de solución, lo que, desde luego es fundamental para poder abordar con éxito situaciones en las que se producen fallos en las aplicaciones integradas. Otra carencia importante está en relación con la cardinalidad de filtros y tareas, que en la mayoría de los casos tan sólo permiten una fuente de mensajes como entrada; esta limitación ha demostrado en la práctica ser problemática dado que conduce a modelos en los que un conjunto de tareas que de forma natural forman parte de un mismo filtro deben ser agrupadas en dos o más filtros diferentes. Estas conclusiones se derivan de nuestra experiencia práctica en los dos proyectos que nos han servido de motivación para el desarrollo de nuestro trabajo. En breve esperamos poner en marcha una nueva experiencia en colaboración con la empresa Sytia Informática, S.L., en esta ocasión en el contexto de un sistema B2BI para la gestión de imágenes médicas.

Referencias

1. C. Chang, M. Kayed, M.R. Girgis, and K.F. Shaalan. Survey of web information extraction systems. *IEEE Transactions on Knowledge and Data Engineering*, 18(10):1411–1428, 2006.
2. B.A. Christudas. *Service Oriented Java Business Integration*. Packt Publishing, 2008.
3. J. Davies, D. Schorow, and D. Rieber. *The Definitive Guide to SOA: Enterprise Service Bus*. Apress, 2008.
4. P. Delia and A. Borg. *Mule 2: Official Developer's Guide to ESB and Integration Platform*. Apress, 2008.
5. Apache Foundation. Apache Camel home. Available at http://activemq.apache.org/camel.
6. M. Fowler. *Patterns of Enterprise Application Architecture*. Addison–Wesley, 2002.
7. G. Hohpe. Your coffee shop does not use two-phase commit. *IEEE Software*, pages 64–66, 2005.
8. G. Hohpe and B. Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley, 2003.
9. iWay Software. iWay adapter for PeopleSoft. Available at http://www.iwaysoftware.com/products/adapters/peoplesoft.html.
10. A. Kleppe, J. Warmer, and W. Bast. *MDA Explained*. Addison–Wesley, 2003.
11. D. Messerschmitt and C. Szyperski. *Software Ecosystem: Understanding an Indispensable Technology and Industry*. MIT Press, 2003.
12. D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern-Oriented Software Architecture*, volume 2. John Wiley, 2000.

13. D. Skeen. A formal model of crash recovery in a distributed system. *IEEE Transactions on Software Engineering*, 9(3):219–228, 1983.
14. Inc. SpringSource. Spring integration home. Available at http://www.springframework.org/spring-integration.
15. W3C. Web services message exchange patterns. Available at http://www.w3.org/2002/ws/cg/2/07/meps.html#id2612442.
16. C. Walls and R. Breidenbach. *Spring in Action*. Manning Publications, 2004.
17. J. Weiss. Aligning relationships: Optimizing the value of strategic outsourcing. Technical report, IBM, 2005.
18. I.H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2005.
19. D. Woolston. *Foundations of BizTalk Server 2006*. Apress, 2007.