Modernización Software mediante Descubrimiento de Vistas de Procesos de Negocio

Ricardo Pérez-Castillo[1], Barbara Weber[2], Ignacio García-Rodríguez de Guzmán[1] and Mario Piattini[1]

[1] Alarcos Research Group, University of Castilla-La Mancha
Paseo de la Universidad, 4 13071, Ciudad Real, Spain
{ricardo.pdelcastillo, ignacio.grodriguez, mario.piattini}@uclm.es
[2] University of Innsbruck
Technikerstraße 21a, 6020 Innsbruck, Austria
barbara.weber@uibk.ac.at

Resumen. Una de las ventajas de la modernización de software frente a la reingeniería tradicional es que permite descubrir y refactorizar modelos a alto nivel de abstracción, como modelos de procesos de negocio. Estos modelos pueden ser usados para modernizar sistemas de información existentes que contienen información de negocio heredado que no puede ser descartado por no encontrarse en ningún otro sitio. La localización de conceptos es clave durante la modernización software para poder emplear de forma selectiva los modelos de procesos de negocio descubiertos. La localización de conceptos mapea los diferentes conceptos (e.g., actividades de negocio, funcionalidades, etc.) en las partes específicas del código fuente que las soportan. Este artículo presenta una técnica para la localización de conceptos basada en el descubrimiento de vistas parciales de procesos de negocio desde modelos de código. La técnica consiste en una transformación de modelos que combina en la entrada un modelo de código (que representa una parte del sistema de información) y un modelo de ejecución (que contiene las actividades de negocio ejecutadas por el sistema tras un periodo en producción). La transformación produce como salida un modelo que representa la porción del proceso de negocio soportado por el modelo de código fuete. Finalmente, se provee la implementación de la transformación así como un caso de estudio para facilitar su adopción en la industria y validar la propuesta.

Palabras Clave: Vistas de Procesos de Negocio, Localización de Conceptos, Modernización Software, Knowledge Discovery Metamodel.

1 Introducción

Los sistemas de información de las organizaciones no es un activo estático que permanece imperturbable a lo largo del tiempo, sino que se van deteriorando a lo largo del tiempo como consecuencia del mantenimiento descontrolado [23]. De esta forma, estos sistemas se convierten en sistemas de información heredados y cuando

sus niveles de calidad caen por debajo de los límites aceptables deben ser reemplazados o modernizados. A fin de mantener el nivel de competitividad, las organizaciones deben implementar nuevas versiones de sus sistemas de información heredados soportando todos los procesos de negocio actuales que realmente llevan a cabo [18]. Sin embargo, los modelos de procesos de negocio de las organizaciones, los cuales describen un conjunto de actividades coordinadas para lograr los objetivos de negocio común [5], están normalmente desactualizados. Esto se debe a que los sistemas heredados van incorporando nuevos procesos de negocio durante el mantenimiento progresivo. Esa nueva información, la cual está solo presente en los sistemas de información heredados, ha de ser extraída a fin de ser preservada durante la modernización de estos sistemas para evitar que se pierda. Así se consigue alinear los sistemas de información heredados y los procesos de negocio reales [3].

La preservación de la información de negocio embebida en los sistemas de información heredados entraña dos desafíos, el primero es el descubrimiento de la información de negocio en sí mismo, y el segundo consiste en el uso eficaz de esa información para poder mejorar la evolución y modernización de los sistemas de información heredados.

El descubrimiento de la información embebida de negocio ha sido tradicionalmente abordado mediante minería de procesos de negocio. La minería de procesos de negocio provee una serie de técnicas y algoritmos que suelen tomar como entrada registros de eventos (representando la ejecución de actividades de negocio) y obtienen los procesos de negocio correspondientes [20]. Estos registros de eventos son obtenidos por la mayoría de sistemas orientados a procesos (e.g., ERP (Enterprise Resource Planning), CRM (Customer Relationship Management), etc.) y son representados en un formato concreto, MXML (Mining XML) [22]. No obstante, los sistemas de información heredados no son tradicionalmente orientados a procesos, es decir, no soportan explícitamente la definición de procesos de negocio y por tanto no registran eventos de las actividades de negocio ejecutadas. Con el objetivo de poder aplicar las técnicas de minería de procesos de negocio a sistemas de información no orientados a procesos se ha desarrollado una técnica para obtener registros de eventos basada en la inserción de trazas en el código fuente heredado [14].

El uso eficaz de los procesos de negocio descubiertos es el segundo desafío para modernizar los sistemas preservando la información de negocio. Todas las funcionalidades y servicios implementados en el sistema de información evolucionado deben dar soporte a los procesos de negocio descubiertos [3]. Para poder realizar esto se necesita tener localizado cada concepto en el código fuente, lo cual es conocido como *localización de conceptos* [2]. La localización de conceptos permite conocer qué partes de código fuente soportan una parte o concepto de un proceso de negocio, o a la inversa qué partes de un proceso de negocio dado son implementadas por una pieza o componente de código fuente.

Este trabajo parte de la técnica propuesta para la obtención de registros de eventos desde sistemas de información heredados [14] aunque se centra en el segundo desafío para la preservación de la información de negocio, su uso eficaz durante la modernización. La contribución principal es una técnica para la localización de conceptos basada en el descubrimiento de vistas de procesos de negocio a partir de un

modelo de registro de eventos y un modelo parcial de código el cual se quiere localizar. Una vista de un modelo de procesos de negocio muestra una representación parcial de acuerdo a alguna propiedad de forma similar a las vistas de tablas en bases de datos [8]. La técnica propuesta permite descubrir los procesos de negocio embebidos en sistemas de información heredados, pero se realiza a través de un conjunto de vistas de procesos generadas por cada parte del código fuente.

La técnica sigue los principios del desarrollo dirigido por modelos por lo que se ha metamodelado tanto los modelos de entrada como salida y se ha automatizado además mediante una transformación de modelos en "Y" para facilitar su adopción y validación.

El resto del artículo se organiza de la siguiente forma. La sección 2 resume el trabajo relacionado. La sección 3 presenta en detalle la técnica propuesta de localización de conceptos basada en la obtención de vistas de procesos de negocio. La sección 4 presenta la validación empírica de esta técnica. Finalmente, la sección 5 discute las conclusiones y trabajo futuro.

2 Trabajo Relacionado

Varios trabajos sobre localización de conceptos durante el mantenimiento software han sido propuestos hasta el momento. *Wilde et al.* [24] proponen una técnica para localizar funcionalidades de usuario en un sistema de información heredado. *Eisenbarth et al.* [2] localizan características en el código fuente recopilando información de un conjunto de escenarios de ejecución que invocan dichas características. *Marcus et al.* [6] mapean conceptos expresados por los ingenieros de mantenimiento en lenguaje natural sobre partes relevantes del código fuente. *Chen et al.* [1] proponen una técnica que analiza el código fuente y obtiene grafos abstractos de dependencia de subsistemas.

El problema de estas técnicas de localización es que expresan los conceptos en diferentes formatos (e.g., como funcionalidades de usuario, grafos de dependencia, etc.). En cambio, el enfoque propuesto considera el estándar *Business Process Modeling and Notation* (BPMN) [10], el cual puede ser entendido tanto por expertos de negocio como ingenieros de mantenimiento. Los conceptos usados son además las actividades dentro de los procesos de negocio, las cuales representan conceptos a alto nivel de abstracción, permitiendo así ventajas durante la fase de reestructuración durante la modernización software [19]. Otro trabajo relacionado que usa este enfoque es el propuesto por *Motahari et al.* [8], el cual obtiene vistas de procesos de negocio aunque lo hace por cada subsistema heredado, por lo que la localización de conceptos de grano fino no se realiza al contrario que la técnica propuesta.

A parte de la localización de conceptos, otros trabajos se han centrado en el descubrimiento de procesos de negocio para preservarlo durante la modernización software. La mayoría de las propuestas proveen técnicas de minería basadas en registros de eventos, las cuales se consideran de tipo dinámico al usar la información de ejecución del sistema. Algunos ejemplos de estas técnicas son el *algoritmo alfa* propuesto por *Van der Aalst et al.* [21] que permite descubrir el flujo de control de los

procesos de negocio desde registros de eventos, o el algoritmo genético mejorado para el descubrimiento de procesos de negocio propuesto por *Madeiros et al.* [7]. Además, *Pérez-Castillo et al.* [14] proponen una técnica para obtener registros de eventos desde sistemas de información no orientados a procesos. Está técnica permite aplicar los algoritmos anteriores para obtener los procesos de negocio soportados por cualquier tipo de sistema de información heredado.

A parte de las técnicas dinámicas, existen técnicas estáticas para el descubrimiento de procesos de negocio, las cuales analizan estructuralmente el código fuente para recuperar la información de negocio embebida. Por ejemplo, *Zou et al* [25] proponen una técnica que analiza el código fuente y aplica una serie de reglas heurísticas para inferir procesos de negocio. *Pérez-Castillo et al.* [13] proponen una técnica similar basada en el desarrollo dirigido por modelos permitiendo analizar otros artefactos software además del código fuente.

La Fig. 1 (a) y (b) compara los enfoques estático y dinámico para descubrir procesos de negocio. Mientras el estático analiza elementos relacionados con el lenguaje de programación, el enfoque dinámico se centra en información que sólo puede ser conocida en tiempo de ejecución como por ejemplo el valor específico de ciertas variables, polimorfismo, etc. Los enfoques estáticos normalmente descubren exhaustivamente grandes procesos de negocio, aunque muchas partes descubiertas son erróneas por lo que no es muy preciso. Por el contrario, los enfoques dinámicos descubren procesos de negocio más compactos pero más precisos. La solución propuesta en la siguiente sección combina ambos enfoques para tomar las ventajas de ambos.

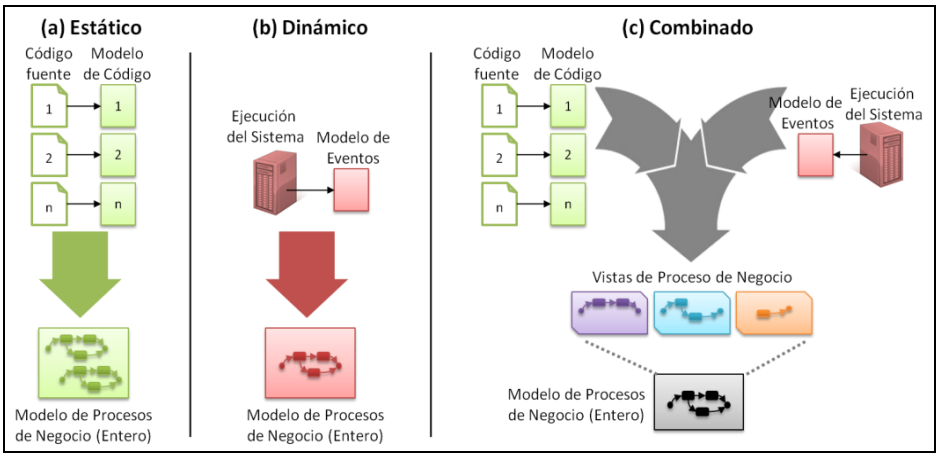

Fig. 1. Comparación de los tres enfoques para el descubrimiento de procesos de negocio.

3 Localización de Conceptos basada en Vistas de Procesos

La técnica de localización de conceptos propuesta se basa en el descubrimiento de vistas de procesos de negocio partiendo desde código fuente. La idea es poder conocer qué parte dentro del proceso de negocio (vista) soporta una pieza concreta de código fuente heredado. Además, cuando se obtiene la vista de proceso de negocio, a cada actividad de negocio dentro de la vista se le enlaza a la unidad de código fuente particular que soporta dicha actividad. Como resultado se obtiene una localización de conceptos bidireccional y de grano fino.

Esta técnica combina el análisis estático de código fuente con el análisis dinámico basado en los registros de eventos (véase Fig. 1 (c)) siguiendo los principios del desarrollo dirigido por modelos. La técnica toma por un lado el modelo de código fuente obtenido estáticamente (cf. sección 3.1), y por otro lado considera el modelo de eventos que representa la ejecución del sistema heredado (cf. sección 3.2). Además, la técnica es automatizada por medio de una transformación en "Y" que produce un modelo de vista de procesos de negocio (cf. sección 3.3).

Ambos modelos de entrada son representados mediante KDM (Knowledge Discovery Metamodel) [4], el cual es un estándar ISO/IEC especialmente pensado para representar modelos desde la información extraída durante la ingeniería inversa de sistemas de información heredados. KDM se ha convertido en el formato común de intercambio de información entre herramientas de ingeniería inversa y modernización de software. El uso de este estándar asegura la aplicabilidad de la técnica en la industria de ingeniería del software.

3.1 Modelo de Código Fuente

El modelo de código es el primer modelo de entrada y ofrece una representación del código fuente de forma independiente del lenguaje de programación. El modelo de código representa los constructores de programa a nivel de implementación así como relaciones entre estos desde un punto de vista estático. Pueden existir varios modelos de código para representar todas las partes de un sistema de información heredado. Puede representarse modelos de código de grano fino como modelos que representan una unidad de compilación, o de grano grueso como modelos por cada subsistema. Los ingenieros de mantenimiento son los que toman la decisión sobre la granularidad de los modelos de código para obtener las vistas de procesos de negocio en cada caso.

Los modelos de código son obtenidos mediante el análisis estático del código fuente, que se basa en técnicas de análisis de la teoría de compiladores que proveen una representación abstracta de la estructura del código fuente. Concretamente, el enfoque propuesto usa la técnica propuesta en un trabajo previo [12], la cual analiza elementos de programa y construye un modelo de código de acuerdo al metamodelo KDM. El modelo es construido considerando las metaclases de los paquetes *code* y *action* definidos por el estándar KDM [4], ya que el metamodelo KDM está dividido en varias capas y paquetes que permiten representar varios artefactos software. La Fig. 2 muestra la parte del metamodelo KDM concerniente a los paquetes *code* y *action*. El elemento *CodeModel* es el elemento raíz y se compone de un conjunto de

AbstractCodeElements, el cual es una metaclase abstracta para representar todos los posibles elementos del código fuente (e.g., *CallableUnit* que representa los métodos, funciones o procedimientos; *StorableUnit* que representa las variables de programa; etc.). Las instancias de la metaclase *CodeElement* están interrelacionadas a su vez por medio de instancias *AbstractCodeRelationship*, una metaclase que representa las diferentes relaciones entre elementos de código (e.g., *Flow* que simboliza el flujo de ejecución normal entre dos sentencias; *Calls* que representa una invocación a una unidad invocable; *Reads* y *Writes*, que representan la lectura o escrituras de datos; etc.).

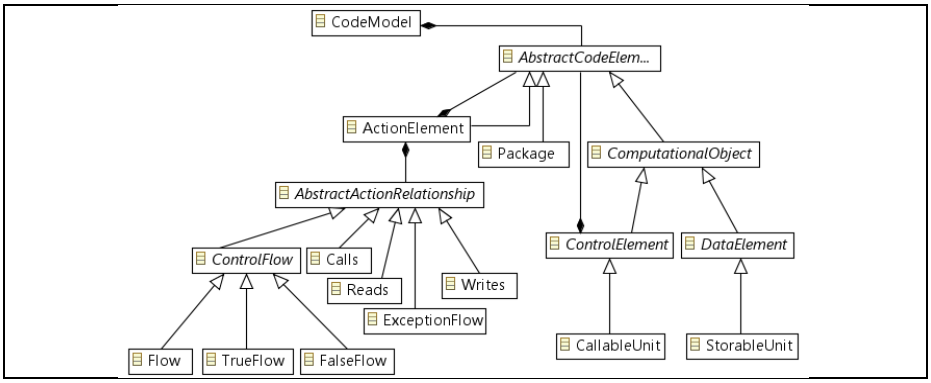

Fig. 2. El metamodelo KDM concerniente al paquete *code* y *action*

3.2 Modelo de Eventos

El modelo de eventos, el segundo modelo de entrada, representa varias secuencias de ejecución de actividades de negocio, las cuales se conocen como instancias de procesos de negocio. Al contrario que el modelo de código, el modelo de eventos describe el sistema de información heredado desde un punto de vista dinámico, es decir, representa la ejecución del sistema.

El modelo de eventos se obtiene durante la ejecución de un sistema de información completo, mientras que los modelos de código pueden ser obtenidos por diferentes partes del sistema. A pesar de ser completo, el modelo de eventos no es exclusivo ya que puede haber varios modelos de eventos concernientes a ejecuciones en periodos de tiempo diferentes. El modelo de eventos tiene el potencial suficiente como para descubrir el proceso de negocio subyacente a todo un sistema de información, aunque se utiliza conjuntamente con modelos de código que permiten filtrar la información para obtener diferentes vistas del proceso de negocio completo.

Como se comentó, la obtención de modelos de eventos es casi directa para sistemas de información orientados a procesos. No obstante, en cualquier otro tipo de sistema sin ningún mecanismo de registro de eventos, el modelo de eventos ha de ser obtenido mediante técnicas de ingeniería inversa. El enfoque propuesto usa la técnica desarrollada en un trabajo previo [14] que consiste en la inserción de trazas en el

código fuente que habilitan al sistema a registrar la ejecución de actividades de negocio que subyacen en diferentes partes del código fuente.

El modelo de eventos es representado en este caso mediante el paquete *event* del metamodelo KDM. La Fig. 3 muestra este metamodelo así como otras metaclases, de otros paquetes de KDM, usadas para representar información adicional necesaria para facilitar la localización de conceptos. *EventModel* es la metaclase raíz y representa el modelo de eventos, el cual contiene a su vez el conjunto de procesos de negocio soportados por un sistema de información que vienen representados mediante un elemento *EventResource* estereotipado con <<Process>>. A su vez, cada proceso contiene un conjunto de instancias de proceso de negocio que también son representadas con un conjunto de *EventResources*, pero en este caso son estereotipados con <<ProcessInstance>>. Además, la metaclase *EventResource* se especializa en las metaclases *State*, *Transition* y *Event*. La metaclase *Event* es la que se emplea para representar un evento de ejecución de una actividad de negocio, la cual representa en su atributo *name* el nombre de la actividad de negocio ejecutada, y en su atributo *kind* si el evento de ejecución fue de inicio (*start*) o de fin de ejecución (*complete*). Cada evento puede contener información adicional acerca del usuario que disparó la ejecución (*originator*) y el momento en el que se produjo el evento (timestamp). Esta información es representada mediante la definición de etiquetas (*TagDefinition*) y valores etiquetados en el evento (*TaggedValue*) (véase Fig. 3). Finalmente, cada instancia de la metaclase *Event* tiene vinculado una instancia de la metaclase *CodeElment* de un modelo de código que indica la pieza de código fuente que da soporte a la ejecución de la actividad de negocio ejecutada durante ese evento (véase Fig. 3).

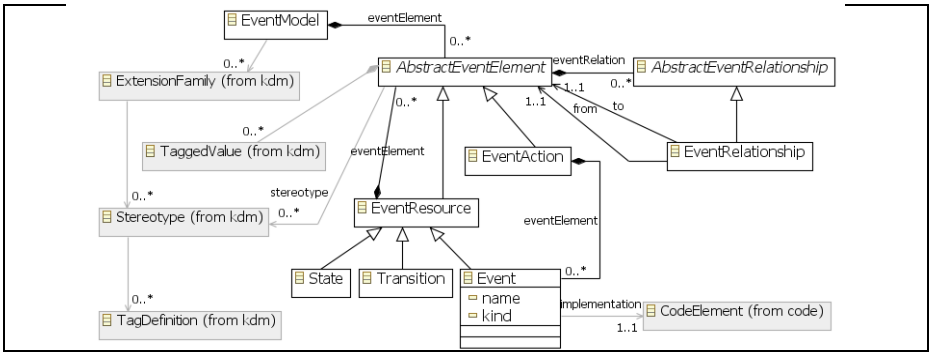

Fig. 3. El metamodelo KDM concerniente al paquete *event*

3.3 Transformación de Modelos para la Generación de Vistas de Procesos

Para obtener las diferentes vistas de procesos de negocio desde diferentes modelos de código se propone una transformación de modelos en "Y" (por su esquema visual). La transformación toma como entrada un modelo de código y un modelo de eventos, y produce como salida un modelo de proceso de negocio (véase Fig. 1 (c)). La transformación descubre procesos de negocio desde el modelo de eventos mediante el

filtrado de actividades de negocio que están presentes en el modelo de código. La transformación se basa en un conjunto de patrones que reconocen estructuras en los modelos de entrada (modelos de eventos y código) y generan otras estructuras predefinidas en el modelo de salida (vistas de procesos de negocio):

Patrón Secuencia. Este patrón crea el flujo básico del proceso de negocio. Para ello, se crea una secuencia de actividades de negocio por cada instancia de proceso de negocio en el modelo de eventos. Además, la anterior secuencia es filtrada respecto a otra secuencia de actividades de negocio que se crea por cada elemento *CallableUnit* en el modelo de código, las cuales son conectadas por flujos de secuencia por cada invocación (representada por elemento *Calls*) que existe entre los elementos *CallableUnit*. De esta manera se filtran las actividades de negocio que se dan sólo en esa parte de código fuente.

Patrón Ramificación. Este patrón obtiene diferentes flujos de actividades paralelos que pueden ejecutarse de forma exclusiva o inclusiva en el proceso de negocio. En primer lugar, se crean el conjunto de posibles ramas paralelas desde el modelo de eventos buscando secuencias alternativas. Para ello se buscan secuencias de actividades con partes comunes y no comunes. Por ejemplo, imagínese un modelo de eventos con las secuencias de actividades {A, B, x, C} y {A, B, y, C}. En este caso se descubriría el proceso de negocio con la secuencia lineal de actividades A y B, una bifurcación que contendría en paralelo las actividades 'x' e 'y', y finalmente la actividad común C. En segundo lugar, estas bifurcaciones serán consideradas cuando en el modelo de código exista un elemento de bifurcación (e.g., estructuras *if-then-else*, *switch*, etc.) en el elemento *CallableUnit* que representa la actividad de negocio correspondiente.

Patrón Objetos de Datos. Este patrón construye un objeto de datos en el proceso de negocio por cada elemento *StorableUnit* en el modelo de código. Además, genera una asociación entre la actividad de negocio (creada desde un elemento *CallableUnit*) y el objeto de datos por cada lectura (elemento *Reads*) o escritura (elemento *Writes*) que tiene el elemento *CallableUnit*. Estos elementos son creados cuando existan las actividades de negocio correspondientes registradas en el modelo de eventos.

Patrón Colaboración. Este patrón obtiene una actividad de negocio y un flujo de secuencia de ida y vuelta desde un elemento *CallableUnit* que invoca a otra unidad externa (e.g., un método de un componente software cuyo código fuente no es accesible y por tanto no pude ser analizado). Como en los anteriores patrones, será creado siempre y cuando la actividad de negocio esté registrada en alguno de los eventos.

Para favorecer la adopción y validación de la transformación propuesta, ésta es implementada usando QVT (Query/View/Transformation) [9]. QVT provee dos lenguajes relacionados aunque diferentes: QVT Operational Mappings (QVTo) de naturaleza imperativa y QVT Relation (QVTr) de naturaleza declarativa. El enfoque propuesto es implementado usando QVTr ya que la definición de los patrones y las restricciones entre los modelos de origen y destino se ve favorecida por lenguajes declarativos. Una transformación de modelos en QVTr consiste en un conjunto de relaciones con dos tipos de dominios, los cuales definen variables que pueden ser encontradas en el tipo de metaclase dado en el dominio [9]. Los dominios de entrada

(etiquetados con *checkonly*) definen un conjunto de metaclases de los modelos de entrada así como un conjunto de restricciones relacionadas con aquellas metaclases que deben ser satisfechas, es decir, aquellas que deben existir en el modelo de entrada. Los dominios de salida (etiquetados con *enforce*) definen una plantilla de metaclases y sus propiedades que deben ser modificadas o creadas en el modelo de salida para satisfacer la relación.

```
1  relation ProcessInstance2Sequence {
2    xTaskName : String;
3    xTask2Name : String;
4    xProcessName : String;
5    checkonly domain eventModel instance : event::EventResource {
6    };
7    checkonly domain eventModel ev1 : event::Event {
8      name = xTaskName
9    };
10   checkonly domain eventModel em : event::EventResource{
11     name = xProcessName,
12     eventElement = inst : event::EventResource {
13       name = instance.name,
14       eventElement = ev2 : event::Event { name = xTask2Name }
15     }
16   };
17   enforce domain bpmn bpd: bpmn::BusinessProcessDiagram  {
18     Pools = p : bpmn::Pool {
19       name = xProcessName,
20       ProcessRef = pr : bpmn::Process {
21         Name = em.name,
22         GraphicalElements = t1 : bpmn::Task {
23           Name = ev.name.substringAfter('.'),
24           process = parent : bpmn::Process { Name = em.name }
25         },
26         GraphicalElements = t2 : bpmn::Task {
27           Name = ev2.name.substringAfter('.'),
28           process = parent : bpmn::Process { Name = em.name }
29         },
30         GraphicalElements = secf1 : bpmn::SequenceFlow {
31           SourceRef = t1,
32           TargetRef = t2,
33           process = parent : bpmn::Process { Name = em.name }
34         }
35       }
36     }
37   };
38   when {
39     subsequent(instance, ev1, ev2)
40   }
41 }
42 query subsequent (proInst:EventResource, ev1:Event, ev2:Event):Boolean {
43   proInst.eventElement.indexOf(ev1)=1+ proInst.eventElement.indexOf(ev2)
44   and event1 <> event2 and event1.name <> event2.name
45 }
```

Fig. 4. Ejemplo de relación QVTr: 'ProcessInstance2Sequence'

Por limitaciones de espacio, se muestra una relación para ilustrar la implementación. La transformación completa puede ser consultada online en [11]. La Fig. 4 muestra la relación *'ProcessInstance2Sequence'*, la cual soporta parcialmente el patrón secuencia. Esta relación define tres dominios *checkonly* (líneas 5 a 16) que son definidos respectivamente sobre instancias de procesos de negocio, eventos, y procesos de negocio. Esta relación es invocada desde otra relación que toma el modelo de código y la pasa, como parámetros, los valores concretos para estos tres

dominios. Estos dominios comprueban la existencia de dos eventos diferentes en una misma instancia de proceso de negocio. Por otra parte, el dominio *enforce* (líneas 17 a 37 en la Fig. 4) crea las dos actividades de negocio respectivas (líneas 22 and 26) así como un flujo de secuencia entre ambas actividades (línea 30). El flujo de secuencia es creado en el modelo de vista de procesos de negocio para aquellas actividades que son contiguas en el modelo de eventos, lo cual se comprueba invocando a la consulta *'subsequent'* (líneas 42 a 45 en la Fig. 4).

4 Caso de Estudio

La técnica propuesta es validada por medio de un caso de estudio que aplica la técnica a un sistema de información real. Los resultados son comparados con los resultados que se obtuvieron en un caso de estudio previo con el mismo sistema de información al que se aplicó el descubrimiento de procesos de negocio completo de acuerdo al enfoque estático. El caso de estudio ha sido planificado y ejecutado siguiendo el protocolo para la realización de casos de estudios en el campo de la ingeniería del software propuesto por *Runeson et al.* [17].

4.1 Diseño

El objeto del estudio es la técnica propuesta para la localización de conceptos basada en el descubrimiento de vistas de procesos de negocio, y el propósito del estudio es la evaluación de su efectividad. La pregunta de investigación es: ¿Puede la técnica propuesta obtener vistas de procesos de negocio, localizadas en el código fuente, con mejor calidad que los procesos de negocio obtenidos mediante técnicas estáticas?

En primer lugar, por cada paquete de código fuente se obtiene un modelo de código. Por otra parte, un modelo de eventos es obtenido mediante la ejecución de una versión instrumentada del sistema de información (mediante trazas insertadas para el registro de eventos). En segundo lugar y para poder contestar a la pregunta de investigación, un conjunto de vistas del proceso de negocio son obtenidas combinando cada modelo de código con el modelo de eventos. Las vistas de proceso de negocio son entonces analizadas para tomar ciertas medidas como: (i) la complejidad, calculada como el número de flujos de secuencia entre el número de actividades de negocio [16]; (ii) la precisión en el descubrimiento de la vista respecto al proceso de negocio del mundo real; (iii) índice de recuperación que indica la completitud en el descubrimiento de la vista respecto del proceso real (es decir, si se quedaron muchas actividades o no sin ser descubiertas); (iv) la media armónica entre precisión e índice de recuperación que combina ambas medidas; y finalmente el tiempo empleado en realizar la transformación entre modelos. El objetivo es comparar los procesos de negocio obtenidos meramente desde modelos de código, y las vistas de proceso de negocio obtenidas combinando los modelos de código y de eventos, y ver si se facilita o no la localización de conceptos.

El estudio toma como caso a un sistema de información heredado real denominado *Villasante Lab*, el cual soporta los procesos de negocio de una empresa dedicada al

análisis químico y bacteriológico de aguas y residuos. Desde un punto de vista tecnológico, el sistema consiste en una aplicación web desarrollada en la plataforma Java, la cual ha sido mantenida durante los últimos cinco años. El tamaño de la aplicación es de 28.8 KLOC (miles de líneas de código fuente).

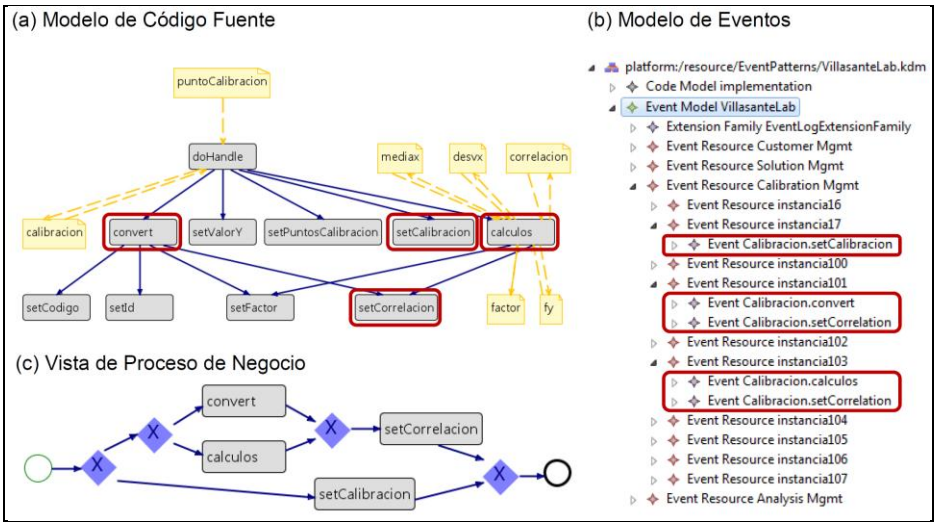

Fig. 5. Modelos de entrada y salida de ejemplo para un paquete de código de *Villasante Lab*

4.2 Ejecución

La Fig. 5 presenta como ejemplo los tres modelos envueltos en la transformación propuesta. La Fig. 5 (a) es la representación gráfica del modelo de código, obtenido desde el paquete de código fuente 'calibration', el cual aglutina un conjunto de métodos, llamadas entre ellos y variables de programa. La Fig. 5 (b) representa el modelo de eventos para todo el sistema *Villasante Lab*. El proceso de negocio 'Calibration Management' está expandido con algunos eventos marcados que coinciden con elementos del modelo de código (compárese Fig. 5 (a) y (b)). Finalmente, la Fig. 5 (c) representa la vista del proceso de negocio que se obtiene tomando como entrada los anteriores modelos de entrada. Esta vista representa la porción del proceso de negocio soportado por el paquete de código fuente 'calibration', por lo que las cuatro actividades de ese modelo de proceso de negocio (es decir, *convert*, *calculos*, *setCorrelation*, *setCalibracion*) quedan localizadas en ese paquete de código fuente.

Después de ejecutar la transformación para todos los modelos de código, la información necesaria para evaluar las medidas propuestas en el diseño del caso de estudio es recogida. La Tabla 1 muestra la media y la desviación estándar de cada una de las medidas tanto para la ejecución de la transformación propuesta que sigue un enfoque combinado, como para los valores obtenidos en la validación de la técnica previa basada solamente en análisis estático del código fuente [12].

Tabla 1. Medidas tomadas para la técnica propuesta (combinada) y la técnica estática

	Enfoque Combinado					Enfoque Estático				
	Complejidad	Precisión	Índice de Recuperación	Media Armónica	Tiempo de Transf. (s)	Complejidad	Precisión	Índice de Recuperación	Media Armónica	Tiempo de Transf. (s)
Media	1.00	0.84	0.77	0.81	1255.5	1.78	0.53	0.84	0.65	2.1
Des. Estd.	0.20	0.09	0.01	0.03	224.4	1.00	0.16	0.08	0.14	1.6

4.3 Análisis e Interpretación

La técnica obtiene varias vistas de procesos de negocio, las cuales pueden solaparse entre ellas, es decir, dos o más vistas podrían contener las mismas actividades de negocio. Sin embargo, el menor tamaño de los modelos de procesos de negocio facilita a los ingenieros de mantenimiento centrarse en partes concretas del proceso de negocio que son soportados por piezas específicas de código fuente. La complejidad de las vistas de procesos de negocio es de media 1.00, la cual es menor que la media de complejidad de 1.78 para procesos de negocio completos (véase Tabla 1). Además, la entendibilidad de las vistas es mejor que la de procesos de negocio enteros ya que ésta es inversamente proporcional a la complejidad [15].

Respecto de las medidas relacionadas con la efectividad en el proceso de descubrimiento (precisión, índice de recuperación y media armónica) el enfoque combinado presenta mejores resultados (véase Tabla 1). La media de precisión para las vistas de procesos de negocio (0.84) es mayor que la precisión del enfoque estático (0.53). Esto se debe a que las vistas se obtienen desde el modelo de eventos (actividades ejecutadas realmente) filtradas por un modelo de código, por lo que se descartan posibles actividades de negocio que son descubiertas por error como ocurre en el enfoque estático que sólo confía en modelos de código. Por esta misma razón el índice de recuperación es ligeramente menor para la técnica propuesta, aunque la media armónica (que permite analizar ambas medidas en conjunto) es mejor para la técnica propuesta que para el enfoque meramente estático, 0.81 frente a 0.65 (véase Tabla 1). La pregunta de investigación puede ser por lo tanto contestada positivamente.

En cuanto al tiempo de transformación, se observa que la técnica propuesta consume más tiempo. No obstante, los tiempos no son directamente comparables ya que en esta técnica se combinan varios modelos de entrada, comparando elemento por elemento, para obtener diferentes vistas de procesos de negocio solapadas; mientras que el enfoque estático obtiene procesos de negocio únicos analizando exclusivamente modelos de código.

5 Conclusiones y Trabajo Futuro

Este artículo ha presentado una técnica para el descubrimiento de vistas de procesos de negocio basada en los principios del desarrollo dirigido por modelos. La técnica

considera de forma combinada modelos de eventos que representan sistemas de información heredados desde una perspectiva dinámica, y modelos de código que representa la estructura estática de los sistemas de información heredados. Las vistas de procesos de negocio obtenidas permiten localizar o mapear conceptos (actividades de negocio o subprocesos) sobre el código fuente heredado. Esta localización facilita que la información embebida de negocio que es descubierta pueda ser usada de forma eficiente durante el proceso de modernización de los sistemas de información heredados a fin de preservarla en las nuevas versiones.

Frente a los procesos de negocio descubiertos por otras técnicas similares, las vistas de procesos de negocio facilitan la modernización de software, ya que por un lado localizan el código fuente que soporta ese subconjunto de actividades de negocio y por otro suponen abstraer la complejidad que supondría manejar un proceso de negocio completo. De esta forma, cuando un conjunto de actividades de negocio son modificadas por la organización, se podría conocer que partes del sistema están sujetas a cambios para realinear el sistema de información de la organización con los procesos de negocio que actualmente lleva a cabo dicha organización.

Por otra parte la técnica conforma estándares internacionales por lo que no es sensible a la alta heterogeneidad que hoy en día existe entre los diferentes sistemas de información dentro de una misma organización. Los modelos de código y eventos, obtenidos desde los sistemas de información heredados por ingeniería inversa, son representados usando el reciente estándar KDM, y las vistas son representadas también de forma estándar de acuerdo a la notación BPMN.

Como trabajo futuro se ha planeado la definición de patrones adicionales para el reconocimiento de aspectos adicionales de los procesos de negocio como bucles complejos, decisiones combinadas en varios niveles con diferentes ramas, etc. Además, la validación será replicada con sistemas de información adicionales.

Agradecimientos
Este trabajo ha sido financiado por el programa FPU y los proyectos de investigación ALTAMIRA (JCCM, PII2I09-0106-2463), MOTERO (JCCM and FEDER, PEII11-0366-9449) y MEDUSAS (CDTI (MICINN), IDI-20090557).

References

[1] Chen, K. and V. Rajlich, Case Study of Feature Location Using Dependence Graph, in 8th International Workshop on Program Comprehension. 2000, IEEE C. S. p. 241.

[2] Eisenbarth, T., R. Koschke, and D. Simon, Locating Features in Source Code. IEEE Trans. Softw. Eng., 2003. 29(3): p. 210-224.

[3] Heuvel, W.-J.v.d., Aligning Modern Business Processes and Legacy Systems: A Component-Based Perspective (Cooperative Information Systems). 2006: The MIT Press.

[4] ISO/IEC, ISO/IEC DIS 19506. Knowledge Discovery Meta-model (KDM), v1.1 (Architecture-Driven Modernization). 2009, ISO/IEC. p. 302.

[5] Jeston, J., J. Nelis, and T. Davenport, Business Process Management: Practical Guidelines to Successful Implementations. 2nd ed. 2008, (Elsevier Ltd.). 469.

[6] Marcus, A., A. Sergeyev, V. Rajlich, and J.I. Maletic, An Information Retrieval Approach to Concept Location in Source Code, in Proceedings of the 11th Working Conference on Reverse Engineering. 2004, IEEE Computer Society. p. 214-223.
[7] Medeiros, A.K., A.J. Weijters, and W.M. Aalst, Genetic process mining: an experimental evaluation. Data Min. Knowl. Discov., 2007. 14(2): p. 245-304.
[8] Motahari, H., B. Benatallah, R. Saint-Paul, F. Casati, and P. Andritsos, Process spaceship: discovering and exploring process views from event logs in data spaces. Proc. VLDB Endow., 2008. 1(2): p. 1412-1415.
[9] OMG, QVT. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. 2008, OMG.
[10] OMG, Business Process Model and Notation (BPMN) 2.0. 2009, Object Management Group. p. 496.
[11] Pérez-Castillo, R. KDM to BPMN Transformation implemented in QVT Relations. 2011 26/04/2011 [cited 2011 02/05/2011]; Available from: http://alarcos.esi.uclm.es/per/rpdelcastillo/modeltransformations/KDM2BPMN.htm.
[12] Pérez-Castillo, R., I. García-Rodríguez de Guzmán, O. Ávila-García, and M. Piattini, Business Process Patterns for Software Archeology, in SAC'10. 2010. p. 165-166.
[13] Pérez-Castillo, R., I. García Rodríguez de Guzmán, M. Piattini, and Á.S. Places, A Case Study on Business Process Recovery using an E-Government System. Software Practice & Experience Journal, 2011: p. In Press.
[14] Pérez-Castillo, R., B. Weber, I. García Rodríguez de Guzmán, and M. Piattini, Process Mining through Dynamic Analysis for Modernizing Legacy Systems. IET Software Journal, 2011: p. In Press.
[15] Reijers, H.A. and J. Mendling, A Study Into the Factors That Influence the Understandability of Business Process Models. IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans, 2010. PP(99): p. 1-14.
[16] Rolón, E., F. Ruiz, F. García, and M. Piattini, Evaluation measures for business process models, in SAC-OE'06. 2006, ACM. p. 1567-1568.
[17] Runeson, P. and M. Höst, Guidelines for conducting and reporting case study research in software engineering. Empirical Softw. Eng., 2009. 14(2): p. 131-164.
[18] Ulrich, W.M., Legacy Systems: Transformation Strategies. 2002: Prentice Hall. 448.
[19] Ulrich, W.M. and P.H. Newcomb, Information Systems Transformation. Architecture Driven Modernization Case Studies. 2010, Burlington, MA: Morgan Kauffman. 429.
[20] van der Aalst, W. and A.J.M.M. Weijters, Process Mining, in Process-aware information systems: bridging people and software through process technology, M. Dumas, W. van der Aalst, and A. Ter Hofstede, Editors. 2005, John Wiley & Sons, Inc. p. 235-255.
[21] van der Aalst, W., T. Weijters, and L. Maruster, Workflow mining: discovering process models from event logs. IEEE Transactions on Knowledge and Data Engineering, 2004. 16(9): p. 1128-1142.
[22] van der Aalst, W.M.P., B.F. van Dongenm, C. Günther, A. Rozinat, H.M.W. Verbeek, and A.J.M.M. Weijters, ProM : the process mining toolkit, in BPM'09. 2009. p. 1-4.
[23] Visaggio, G., Ageing of a data-intensive legacy system: symptoms and remedies. Journal of Software Maintenance, 2001. 13(5): p. 281-308.
[24] Wilde, N. and M.C. Scully, Software reconnaissance: mapping program features to code. Journal of Software Maintenance, 1995. 7(1): p. 49-62.
[25] Zou, Y. and M. Hung, An Approach for Extracting Workflows from E-Commerce Applications, in 14th International Conference on Program Comprehension. 2006, IEEE Computer Society. p. 127-136.