Diagnosis de Errores en la Gestión de Procesos Software con Programación con Restricciones

Diana Borrego
ETS Ingeniería Informática
Univ. de Sevilla
dianabn@us.es

María Teresa Gómez-López
ETS Ingeniería Informática
Univ. de Sevilla
maytegomez@us.es

Rafael Martínez Gasca
ETS Ingeniería Informática
Univ. de Sevilla
gasca@us.es

Luisa Parody
ETS Ingeniería Informática
Univ. de Sevilla
lparody@us.es

Resumen

El desarrollo de proyectos software similares es habitual en las empresas de desarrollo de software. Esto proporciona una base de conocimiento sobre cómo se han comportado otros proyectos similares y la posibilidad de detectar el origen de sus deficiencias. En este trabajo se propone enriquecer las reglas de negocio de un proceso software como reglas de cumplimiento del contrato que describe el comportamiento del mismo. Esta descripción del comportamiento se hará utilizando el paradigma de la programación con restricciones. Analizando cada una de las instancias y adaptando las técnicas de diagnosis basada en modelos, podremos detectar qué actividad es incorrecta y tomar las decisiones oportunas para que el fallo no vuelva a ocurrir.

Para llevar a cabo la diagnosis se ha tenido en cuenta que en los procesos de negocio no todas las tareas tienen que estar involucradas en una determinada instancia. Para la diagnosis se han combinado resolución de problemas de satisfacción de restricciones y algoritmos para una diagnosis más eficiente.

1. Introducción

Hoy en día existen multitud de organizaciones de desarrollo de productos software para clientes de diversa índole. Dentro de dichas organizaciones, distintos departamentos trabajan de forma colaborativa para lograr la consecución de los proyectos contratados [9], realizando una correcta gestión de recursos y personal para obtener beneficios. Esto es lo que se conoce como Gestión de proyectos, y a veces condiciona el éxito o fracaso del proyecto en cuanto a los beneficios obtenidos del mismo [12]. Este es el punto de vista de la Ingeniería de los Procesos Software, un área de conocimiento del campo de la Ingeniería del Software que se orienta a la aplicación de métodos y modelos para conocer, racionalizar y en último extremo mejorar la gestión de las actividades que forman parte del ciclo de vida de un producto software. Desde el punto de vista que nos interesa, se trata de una aplicación a la Ingeniería del Software del paradigma de la reingeniería de procesos y de la mejora continua mediante la diagnosis de fallos [1].

En el proceso de contratación, la organización capta pedidos a través de su acción comercial, que se convierten en proyectos a desarrollar e implementar. Antes de comenzar el desarrollo, la organización acuerda con

el cliente el precio del producto software contratado, de forma que este dato es conocido a priori.

A partir de dichos ingresos a recibir, el departamento de negocio determina el beneficio que se espera obtener, así como los porcentajes de costo que se dedicarán a cada fase del proyecto en función de su capacidad, de los compromisos adquiridos con los clientes y de la carga global de trabajo. Con esta información, el departamento técnico debe tomar decisiones acerca de la planificación del proyecto y los recursos necesarios para su consecución con éxito. Dichos recursos engloban tanto al personal como posibles adquisiciones de hardware y software, cursos de formación, etc.

Una vez que el proyecto finaliza, y debido a las decisiones tomadas durante su ejecución, pueden producirse desviaciones de coste con respecto a las estimaciones realizadas por el departamento de negocio. Las causas de estas desviaciones deberían ser conocidas y comunicadas a los distintos departamentos involucrados en el proyecto para poder tomar medidas en próximos proyectos, evitando nuevas desviaciones que mermen los beneficios de la organización.

En este artículo se proponen varias técnicas para diagnosticar las causas de las desviaciones de coste. Dicha diagnosis se realiza a partir de los datos referentes a las decisiones tomadas en proyectos finalizados, como el número de personas o la cantidad de horas dedicadas a una tarea. En base a estos datos, ante la existencia de una desviación de coste, nuestras propuestas localizan la causa del problema para así ayudar a la toma de decisiones en posteriores proyectos de la misma índole.

El enfoque de este trabajo se dirige a organizaciones dedicadas al desarrollo, implantación y mantenimiento de un tipo específico de proyectos que presentan un perfil determinado, con las mismas fases de ejecución, que se pueden englobar en un mismo proceso de negocio cuyas actividades son las distintas fases del proceso software para ese tipo de proyectos. Se tiene información de varias instancias para proyectos similares, y por lo tanto mediante la detección y diagnosis de actividades incorrectas se pueden mejorar ejecuciones futuras [11]. El departamento de negocio determina el contrato a cumplir en cuanto a costes mediante reglas de negocio vinculadas al proceso.

Un proceso de negocio consiste en un conjunto de actividades que se ejecutan coordinadamente dentro de un entorno técnico y organizacional [14]. La base de la gestión de procesos de negocio es la representación explícita de procesos de negocio con sus actividades y el contrato entre ellas. Es posible describir el comportamiento y semántica de los datos manejados en un proceso de negocio mediante el uso de reglas de negocio, que pueden utilizarse para describir el contrato a cumplir por el proceso completo. Ya que cada organización define su propia política sobre cosas como precios, costes, número de empleados, fechas de entrega, etc., es posible especificar un conjunto de reglas diferente para cada proceso de negocio, o para diferentes actividades dentro de un proceso. En este artículo, las reglas de negocio estudiadas definen el contrato en cuanto a ingresos y costes, de forma que determinan los beneficios a conseguir con cada proyecto desarrollado.

Existen trabajos relacionados con el cumplimiento de reglas de negocio asociadas a un proceso [10] [8] [4]. Sin embargo, la cuestión más importante planteada en este artículo es si es posible inferir dónde ha ocurrido un fallo con respecto a la política de negocio y las decisiones tomadas por la organización. Esto sugiere una adaptación de la diagnosis basada en modelos clásica. Para ello, proponemos un framework que detecta, de forma automática, el incumplimiento del contrato establecido mediante reglas de negocio, realizando una diagnosis automática.

Para realizar dicha diagnosis automáticamente, las reglas de negocio serán presentadas como restricciones. Estas restricciones son ecuaciones o inecuaciones sobre variables del proceso, lineales o polinómicas, relacionadas mediante combinaciones booleanas (*and/or*), cuyos valores enteros o flotantes están definidos en un dominio.

Mediante el uso de restricciones la informa-

ción se representa en un nivel más abstracto, ya que los lenguajes basados en restricciones incluyen y mejoran toda la capacidad de representación de las actuales propuestas en el campo de los Sistemas de Gestión de Reglas de Negocio, como Drools, Fair Isaac Blaze Advisor, ILOG JRules and Jess. Asimismo, la representación de las reglas de negocio como restricciones puede proporcionarnos la automatización de la diagnosis.

La estructura del artículo es como sigue: la Sección 2 presenta la notación y las definiciones formales necesarias para la explicación de la propuesta. En la Sección 3 presentamos la especificación de un ejemplo motivante que consta de un proceso de negocio y sus correspondientes reglas de negocio. Un framework para automatizar la diagnosis se presenta en la Sección 4. La Sección 5 contiene la contribución más importante del trabajo, dos propuestas de diagnosis para encontrar las causas de posibles desviaciones en el coste de un proyecto software. La Sección 6 presenta los resultados experimentales, y la Sección 7 concluye el artículo e indica futuras líneas de investigación.

2. Definiciones formales y notación

El propósito de esta sección es presentar algunas nociones básicas de la teoría de procesos de negocio para aclarar conceptos necesarios en secciones posteriores.

Definición 1. Restricción de cumplimiento del negocio (RCN). Es la representación de una regla de negocio como una restricción. Existen dos tipos:

- Restricciones de cumplimiento globales: restricciones que involucran a todas las actividades del proceso de negocio.
- Restricciones de cumplimiento locales: restricciones que involucran sólo a algunas actividades del proceso de negocio.

Debido al importante número de herramientas de resolución eficiente de problemas de satisfacción de restricciones (*Constraint Satisfaction Problem, CSP*), y para automatizar el proceso de diagnosis, se construye un *CSP* a partir de las *RCN* y las variables implicadas en ellas.

Un *CSP* viene definido por un conjunto de variables, dominios y restricciones. Formalmente, un *CSP* está definido por una tupla $<X, D, C>$ donde $X = \{x_1, x_2, \ldots, x_n\}$ es un conjunto finito de variables, $D = \{d(x_1), d(x_2), \ldots, d(x_n)\}$ es un conjunto de dominios de los valores de las variables, y $C = \{C_1, C_2, \ldots, C_m\}$ es un conjunto de restricciones. Una restricción $C_i = (V_i, R_i)$ especifica los posibles valores de las variables en V simultáneamente para satisfacer R.

En el *CSP* construido, los dominios de las variables dependen o bien del contrato definido o bien de la especificación proporcionada.

Las diferentes *RCN* están relacionadas con las distintas actividades del proceso de negocio. Esta relación puede hacerse de forma automática, ya que depende de las variables que participan en cada actividad y restricción.

Definición 2. Mapeo Restricción de cumplimiento/Actividad. Asociación en la que una *RCN* está relacionada con un conjunto de actividades del proceso de negocio.

Para automatizar el proceso de diagnosis, es necesario definir los siguientes conceptos:

Definición 3. Restricción de satisfacción concretada. Es una *RCN* asociada a un valor de verdad.

Definición 4. Mínimo subconjunto insatisfactible (*Minimal Unsatisfiable Subset, MUS*). Dado un conjunto de *RCN* C, un *MUS* de C es un subconjunto de C que es (1) insatisfactible y (2) mínimo, de tal manera que eliminando cualquiera de sus elementos, el resto de elementos en el *MUS* es satisfiable. Es decir, m es un *MUS* de C sii $m \subseteq C : m$ es insatisfactible, y $\forall c \in m, m \backslash \{c\}$ se satisface.

3. Ejemplo motivante

A continuación se presenta un ejemplo de proceso de negocio que engloba las fases de desarrollo e implantación de un producto software. El proceso comienza con la fase de análisis, recogida en una actividad, y termina cuando el producto ha sido implantado en el cliente. Du-

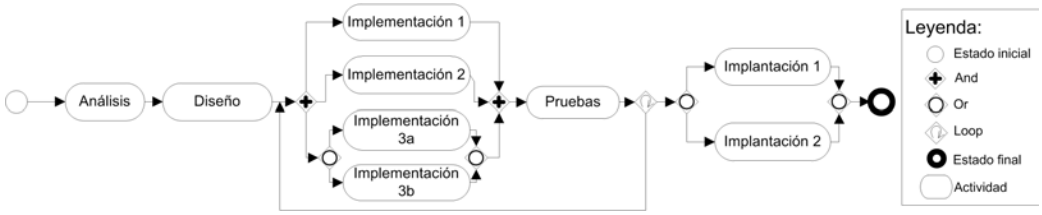


Figura 1: Ejemplo Motivante

rante el proceso se utilizan recursos que conllevan un coste dependiente del tiempo de utilización y el tipo de recurso. Para simplificar el ejemplo sólo se consideran los recursos de personal, con distintos costes dependiendo de distintas categorías.

La Figura 1 muestra el ejemplo en notación BPMN. De hecho, este ejemplo está basado en un ejemplo de diagrama BPMN de la especificación BPMN 1.1 (OMG 2008).

El proceso en la Figura 1 presenta tres tipos de operadores de control de flujo (*or split/join*, *and split/join* y *sequence flow looping*). Estos operadores permiten que ciertas actividades sean ejecutadas en función, por ejemplo, de los ingresos totales por el producto, o del resultado de las pruebas realizadas.

En la Figura 2 se muestran las variables, con sus respectivos dominios, y las *RCN* que serán utilizadas para validar y diagnosticar cada instancia del proceso.

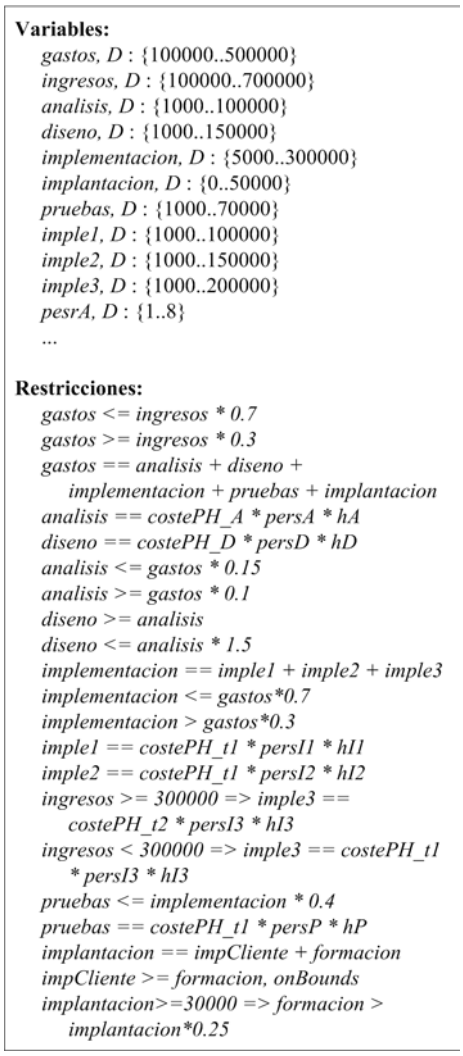
Las variables son: *ingresos* es el total pagado por el cliente; *gastos* es la suma total de los gastos de elaboración e implantación del producto; *analisis* es el coste de la fase de análisis; *diseno* es el coste de la fase de diseño; *implementacion* es el coste de la fase de implementación, que presenta tres subfases *imple*1, *imple*2, *imple*3; *pruebas* es el coste derivado de la fase de pruebas; *implantacion* es el coste de la fase de implantación, que incluye la implantación en cliente *impCliente* y la fase de formación $formacion$; y por último, $persX$ y hX representan el número de personas que participan en cada fase, así como el número de horas de trabajo en esa fase. Asimismo, hay valores constantes que participan en las restricciones, como el coste por persona y hora de los distintos tipos de personal que interviene en el proceso ($costePH\_X$).

A la hora de desarrollar un producto software, las actividades de este proceso de negocio se ejecutan. No obstante, es posible que, tras la ejecución del proceso, y debido a los valores asignados a las variables del *CSP* (por ejemplo, la cantidad de horas y personas empleadas en la fase de diseño), las *RCN* no se satisfagan. En este caso, es necesario llevar a cabo un proceso de diagnosis para determinar las causas de este incumplimiento de contrato.

4. Framework para la diagnosis basada en contrato de procesos de negocio

Para determinar el componente que no funciona correctamente en la diagnosis clásica, basada en modelos, es necesario comparar el modelo del sistema con las variables observables. En el area de procesos de negocio, los detalles del proceso pueden no estar disponibles debido a razones de privacidad, pero deben ser conocidas tanto la política de la empresa como las reglas de negocio que debe satisfacer el flujo de datos. Esto permite al usuario diagnosticar posibles fallos y entender por qué una una instancia en particular no funciona correctamente. No obstante, determinar la causa de un fallo en un gran reto, ya que deben tenerse en cuenta la topología del proceso de negocio y las actividades que participan en cada instancia.

En este artículo se propone un framework basado en el análisis de las *RCN* y los datos de un proceso de negocio para diagnosticar una instancia. El modelo del proceso de negocio y



Variables:

gastos, D : {100000..500000}
ingresos, D : {100000..700000}
analisis, D : {1000..100000}
diseno, D : {1000..150000}
implementacion, D : {5000..300000}
implantacion, D : {0..50000}
pruebas, D : {1000..70000}
imple1, D : {1000..100000}
imple2, D : {1000..150000}
imple3, D : {1000..200000}
pesrA, D : {1..8}
...

Restricciones:

*gastos <= ingresos * 0.7*
*gastos >= ingresos * 0.3*
gastos == analisis + diseno + implementacion + pruebas + implantacion
*analisis == costePH_A * persA * hA*
*diseno == costePH_D * persD * hD*
*analisis <= gastos * 0.15*
*analisis >= gastos * 0.1*
diseno >= analisis
*diseno <= analisis * 1.5*
implementacion == imple1 + imple2 + imple3
*implementacion <= gastos*0.7*
*implementacion > gastos*0.3*
*imple1 == costePH_t1 * persI1 * hI1*
*imple2 == costePH_t1 * persI2 * hI2*
*ingresos >= 300000 => imple3 == costePH_t2 * persI3 * hI3*
*ingresos < 300000 => imple3 == costePH_t1 * persI3 * hI3*
*pruebas <= implementacion * 0.4*
*pruebas == costePH_t1 * persP * hP*
implantacion == impCliente + formacion
impCliente >= formacion, onBounds
*implantacion>=30000 => formacion > implantacion*0.25*

Figura 2: Ejemplo de restricciones de cumplimiento

la diagnosis se basan en las *RCN*.

El framework propuesto es una extensión del clásico *Process Aware Information System (PAIS) framework*. En general, la arquitectura *PAIS* puede verse como el sistema de cuatro niveles presentado en [13], donde de arriba a abajo las capas son: Presentación, Proceso, Aplicación y Persistencia. Como característica fundamental, *PAIS* proporciona los medios para separar el proceso lógico del código de la aplicación. *PAIS* orquesta los procesos en tiempo de ejecución de acuerdo con la lógica definida, y coordina el procesamiento de aplicaciones relevantes y otros recursos. El nuevo framework, mostrado en la Figura 3, presenta una nueva capa donde se lleva a cabo la diagnosis del proceso de negocio. Asimismo, la capa de presentación cuenta con un cuadro de mandos que permite la participación del usuario para introducir datos concretos del mismo modo que la diagnosis clásica usa el modelo observacional de los sensores.

La capa de diagnosis de diagnosis y la capa de proceso son paralelas e independientes, ya que pueden ser ejecutadas en máquinas diferentes, por diferentes actores, y al mismo tiempo.

La Figura 3 muestra una visión general de nuestro framework. Los procesos se modelan en términos de un lenguaje típico de modelado, incluyendo nodos de tarea (las actividades del proceso) así como *and* y *or splits/joins* para modelar el control de flujo. Dicho modelo especifica sólo qué secuencias de actividades pueden ocurrir; no puede modelar dependencias más sutiles o indirectas entre las actividades, debido a las reglas de negocio.

Las restricciones de cumplimiento de negocio se comprueban sobre los estados lógicos que pueden atravesar el proceso. Una forma simple de chequear el cumplimiento es por lo tanto enumerar todos esos estados. Claramente, dado que el número de estados es (en general) exponencial con respecto al tamaño del proceso, dicha propuesta es indeseable. Un modelador humano no toleraría largos tiempos de espera durante el proceso de modelado, y el chequeo del cumplimiento de un repositorio completo de procesos frente a una base de restricciones alteradas puede llegar a ser completamente inviable si cada proceso simple involucra una enumeración de estados.

A continuación se detalla la nueva capa incorporada al framework.

4.1. Capa de diagnosis

Cuando se analiza el modelo de un proceso de negocio con restricciones de cumplimiento,

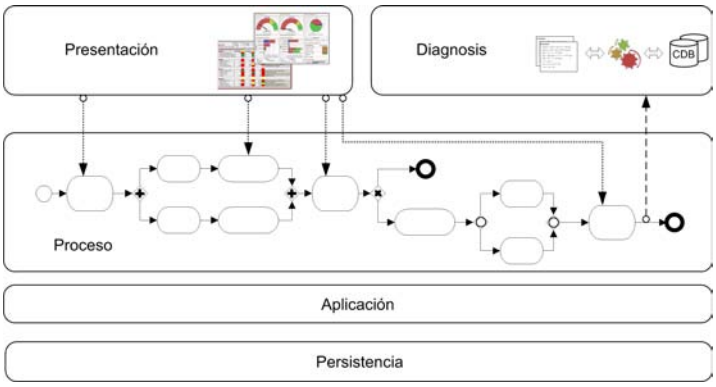


Figura 3: Framework propuesto

se pueden identificar algunas inconsistencias. Para determinarlas, son necesarios tres tipos de información sobre la instancia del proceso de negocio:

- *RCN* que deben cumplirse para esa instancia concreta del proceso.
- Las actividades que han participado en la instancia del proceso, obtenidas a partir de un fichero de log, ya que las condiciones asociadas a los operadores de control de flujo pueden determinar distintos caminos de ejecución.
- Los valores de las variables del flujo de datos que han participado en la instancia. Son aquellas variables que pueden ser instanciadas durante el proceso.

Con esta información, la capa de diagnosis ejecuta tres pasos:

1. Seleccionar qué *RCN* deben ser cumplidas para la instancia determinada.
2. Detectar si el proceso de negocio se ha ejecutado correctamente, cumpliendo en correspondiente contrato indicado mediante las reglas de negocio. Este paso se lleva a cabo construyendo un *CSP* con las *RCN* involucradas en la instancia.
3. Si se detecta una inconsistencia, el proceso de diagnosis determina qué restricción y, por consiguiente, qué variables son responsables de ese no cumplimiento.

Para llevar a cabo la anteriormente mencionada determinación de inconsistencias, y la posterior diagnosis, las *RCN* y el flujo de datos se mapean en un *CSP*. De esta forma, la tarea de diagnosis puede ser realizada usando dos técnicas propuestas, detalladas en la Sección 5.

La capa de diagnosis debe almacenar todas las *RCN*, y obtenerlas posteriormente de una forma eficiente para determinar qué *RCN* deben participar en el proceso. Ya que las reglas de negocio se representan como restricciones, usaremos bases de datos de restricciones (*Constraint Databases, CDBs*) para soportar y manejar dichas reglas.

Cuando se maneja una gran cantidad de datos, la utilización de una base de datos es una decisión obligatoria, especialmente cuando no todas las reglas de negocio se establecen para el proceso completo. El almacenamiento de las reglas de negocio implica también almacenar todos los detalles relativos a sus variables y dominios. Estos tipos de información y reglas de negocio expresadas como restricciones son admitidas en Sistemas de Gestión de Bases de Datos de Restricciones (*Constraint Database Management Systems, CDBMS*).

Las *CDBs* se desarrollan inicialmente en 1992 [7]. La idea básica tras el modelo CDB

es generalizar la noción de una tupla es una base de datos relacional a una conjunción de constraints, ya que una tupla en álgebra relacional puede representarse como una restricción de igualdad entre un atributo de la base de datos y una constante. En un proceso de negocio real, deben ser definidas una gran cantidad de reglas de negocio, por lo que se necesita un repositorio para evaluar estas reglas lo más pronto posible. La CDb usada en este artículo se basa en la *Labelled Object-Relational Constraint Database Architecture (LORCDB Architecture)* [5].

5. Diagnosis basada en contrato de procesos de negocio

La sección previa presenta un framework propuesto compuesto por varias capas. La capa de diagnosis, añadida a dicho framework para conseguir el objetivo de este artículo, hace uso de dos estrategias para proporcionar dos soluciones de diagnosis diferentes, ambas basadas en el paradigma de Programación con Restricciones. Debido a los diferentes operadores de control de flujo que forman la estructura de un proceso de negocio, es necesario tener en cuenta la instancia que ha sido ejecutada para determinar las restricciones que compondrán el *CSP* a resolver, del mismo modo que los datos de entrada del proceso. Es decir, la información acerca de los datos de entrada y las actividades que han sido ejecutadas es de interés ya que definirán qué *RCN* deben cumplirse en cada instancia y si esas restricciones se satisfacen o no dependiendo de los valores de entrada.

Por lo tanto, la capa de diagnosis del framework recibe información acerca de las actividades que han sido ejecutadas para construir el *CSP* correcto. Los operadores de control de flujo que influyen en la traza de ejecución son:

- **Parallel Split (AND).** Este es un hilo de ejecución simple que se divide en dos o más ramas paralelas que se ejecutan concurrentemente. Ya que todas las ramas se ejecutan, si la ejecución del proceso de negocio llega hasta esta estructura, todas las actividades, y por consiguiente todas *RCN* asociadas a ellas, serán tenidas en cuenta en el proceso de diagnosis.

- **Multi-choice (OR/XOR).** Esta es una rama simple que se separa en dos o más ramas. Cuando la ejecución de la rama simple termina, el hilo de control pasa a uno o más ramas dependiendo de la evaluación de una expresión lógica. Por lo tango, el *CSP* a resolver estará compuesto sólo por esas *RCN* asociadas a las actividades e las ramas ejecutadas.

- **Loop.** Esta estructura permite que un conjunto de actividades sean ejecutadas al menos una vez dependiendo de una condición lógica. Con respecto a la incorporación de *RCN* a nuestro *CSP*, es necesario tener en cuenta que las actividades que se encuentran dentro del ciclo pueden ejecutarse varias veces, por lo que las restricciones que afectan a esa parte del proceso de negocio serán instanciadas acorde a todas las ejecuciones realizadas.

Una vez que las restricciones del *CSP* han sido determinadas de acuerdo a la traza del proceso de negocio, entonces la diagnosis puede realizarse con cualquiera de estas dos soluciones propuestas:

- **Diagnosis usando restricciones concretadas.** Esta propuesta tiene en cuenta la relación entre restricciones y actividades. Cada *RCN* es convertida en una restricción concretada al añadirle un valor de verdad, dando lugar a un *CSP* sobrerestringido. Usando las restricciones concretadas minimizaremos el número de restricciones no satisfechas para encontrar la mínima diagnosis.

- **Diagnosis basada en la obtención de los MUSes.** La idea de esta propuesta es encontrar los mínimos subconjuntos inconsistentes de restricciones (*Minimal Unsatisfiable Subsets of constraints, MUSes*, [2]) para encontrar qué es incorrecto en una instancia de un *CSP* sobrerestringido. Mediante la obtención de estos subconjuntos, y calculando sus conjuntos

conflictivos mínimos, podemos encontrar las reglas de negocio que causan la inconsistencia, obteniendo na mínima diagnosis.

Ambas propuestas de diagnosis se detallan en las siguientes subsecciones.

5.1. Diagnosis usando restricciones concretadas

Tal y como se mencionó anteriormente, cada *RCN* da lugar a una restricción concretada al añadirle a la misma su valor de verdad. Para ello, cada restricción cuenta con una variable booleana (*true/false*) que se le asocia mediante una igualdad.

Aplicando esta idea al ejemplo de la Figura 1 se necesitan 21 nuevas variables booleanas, una para cada restricción ($C1$, $C2$, ...). Algunas de las restricciones concretadas que se obtienen se muestran en la Figura 4.

Restricciones concretadas:
*C1 == (gastos<=ingresos*0.7)*
*C2 == (gastos>=ingresos*0.3)*
C3 == (gastosb == analisis + diseno +
implementacion + pruebas + implantacion)
*C4 == (analisis==costePH_A*persA*hA)*
*C5 == (diseno==costePH_D*persD*hD)*
*C6 == (analisis<=gastos*0.15)*
*C7 == (analisis>=gastos*0.1)*
C8 == (diseno>=analisis)
...

Figura 4: Restricciones concretadas

Una vez que las nuevas restricciones están preparadas, el *CSP* a resolver se completa con una función objetivo cuyo propósito es minimizar el número de restricciones no satisfechas, minimizando el número de variables booleanas instanciadas a *false*.

Y como último paso del proceso, debemos conocer las variables instanciadas que son responsables de la inconsistencia. Cada restricción cuenta con varias variables, que podemos agrupar en un conjunto, debiendo encontrar la mínima cantidad de variables que cubra todos esos conjuntos. De acuerdo con la teoría de diagnosis, la mejor forma de encontrar ese cubrimiento mínimo es calculando los conjuntos conflictivos mínimos, cuya definición en la siguiente:

Definición 5. Conjunto conflictivo (CC) para una colección de componentes $\mathcal{C}$ es un conjunto de componentes $\mathcal{H} \subseteq \bigcup_{\mathcal{S} \in \mathcal{C}} \mathcal{S}$ de forma que $\mathcal{H}$ contiene al menos un elemento de cada $\mathcal{S} \in \mathcal{C}$. Un CC de $\mathcal{C}$ es mínimo sii ningún subconjunto de $\mathcal{C}$ es un CC de $\mathcal{C}$. Los mínimos CCs para un conjunto de conjuntos está formado por $\{\mathcal{H}_1, \mathcal{H}_2, \ldots \mathcal{H}_n\}$, donde $\mathcal{H}_i$ es un mínimo CC de componentes. La cardinalidad de $\mathcal{H}_i$ ($|\mathcal{H}_i|$) es el número de componentes de $\mathcal{H}_i$. Esta definición puede ser adaptada a variables o restricciones en lugar de componentes.

Esta propuesta presenta una complejidad computacional despreciable.

5.2. Determinación de MUSes para la diagnosis basada en contrato

Esta implementación está basada en la obtención de todos los *MUSes*, que representan la explicación más concisa de inviabilidad, en términos de número de restricciones involucradas. De hecho, cuando comprobamos la consistencia de un *CSP*, es preferible conocer qué restricciones se contradicen en lugar de sólo saber que el *CSP* es inconsistente.

Debido a la complejidad computacional de la obtención de los *MUSes*, algunas propuestas existentes sólo obtienen algunos de ellos y no todos los *MUSes* de un *CSP* sobrerestringido. Nuestra propuesta se basa en una mejora en [3], la cual está basada en análisis estructural para determinar todo los *MUSes* de forma eficiente. En el citado artículo son presentadas varias técnicas, las cuales mejoran la técnica de distintas formas dependiendo de la estructura de la red de restricciones. Estas técnicas hacen uso del fuerte concepto de entramado estructural de restricciones y de análisis estructural basado en vecindad para mejorar la eficiencia de los algoritmos exhaustivos. Ya que los métodos sistemáticos para resolver problemas combinatorios requieren una alta complejidad computacional, el análisis estructural ofrece una propuesta alternativa para la generación rápida de todos los *MUSes*. Por con-

siguiente, estudios experimentales de estas técnicas superan a la mejor técnica exhaustiva ya que evitan la necesidad de resolver un elevado número de *CSPs* con complejidad exponencial. No obstante ellos añaden ciertos procedimientos nuevos con complejidad polinomial.

En este artículo se aplican dos técnicas, de forma que su complejidad computacional pueda ser comparada:

- **Técnica exhaustiva.** Obtiene los *MUSes* mediante una cola y una lista para almacenar los subconjuntos que satisfacen y no satisfacen las *RCN*, respectivamente (Algoritmo 1).

```
//Siendo C las RCN que presentan la
  inconsistencia:
Q := cola inicializada con las
  restricciones en C
MUS := lista que contendrá los MUSes,
  inicialmente vacía
while(Q no está vacía)
  {c_i ... c_j} := Q.poll() //devuelve y
    elimina la cabeza de Q
  for(c_k ∈ {c_{j+1}..c_n})
    if(NO ∃SubSet^{1...n-1}_{{c_i...c_j}} ∪ c_k ∈ MUS //n
      es la cardinalidad de {c_i ... c_j})
      if({c_i ... c_j} ∪ c_k es satisfiable)
        Q.add({c_i ... c_j} ∪ c_k)
      else
        MUS.add({c_i ... c_j} ∪ c_k)
      endif
    endif
  eldfor
endwhile
```

Algoritmo 1: Algoritmo para obtener los *MUSes* (I)

- **Técnica basada en la vecindad de variables.** Utiliza el concepto de vecindad de variables (*Variable-Based Neighbourhood*) explicado en [3]. Con este fin, esta técnica relaciona *RCN* entre sí como vecinas si comparten ciertas variables no observables, que son aquellas variables cuyos valores permanecen desconocidos hasta el fin de la ejecución del proceso ya que ellas no son determinadas previamente.

El Algoritmo 2 muestra los detalles del proceso.

```
//Siendo C las RCN que presentan la
  inconsistencia:
Q := cola inicializada con las
  restricciones en C
MUS := lista que contendrá los MUSes,
  inicialmente vacía
while(Q no está vacía)
  {c_i ... c_j} := Q.poll() //devuelve y
    elimina la cabeza de Q
  vecinos := expandir({c_i ... c_j})
    //genera los vecinos
  for(c_k ∈ vecinos)
    if({c_i ... c_j} ∪ c_k es satisfiable)
      Q.add({c_i ... c_j} ∪ c_k)
    else
      MUS.add({c_i ... c_j} ∪ c_k)
    endif
  eldfor
endwhile
```

Algoritmo 2: Algoritmo para obtener los *MUSes* (II)

Como mejora sobre la técnica presentada en [3], cuando son calculados los vecinos de un conjunto de restricciones, no todos los vecinos se generan: se establece un orden entre las restricciones, de forma que sólo se generan los vecinos que son posteriores, en orden, a todas las restricciones del conjunto. De esta forma se consigue generar sólo *nuevos* vecinos, evitando el estudio de la consistencia de la misma colección de restricciones repetidamente.

Una vez que se han obtenido todos los *MUSes*, el último paso es conocer qué restricciones y, por lo tanto, qué variables son responsables de la inconsistencia del contrato. Para ello, y al igual que el la técnica anterior basada en restricciones concretadas, utilizaremos el concepto de conjuntos conflictivos mínimos para encontrar el mínimo número de restricciones que cubre a todos los *MUSes* encontrados. El cálculo de estos conjuntos conflictivos nos proporcionará la diagnosis mínima para una instancia del proceso de negocio.

6. Resultados experimentales

En esta sección se presentan los resultados obtenidos utilizando ambas técnicas. Para ello, y debido a que la complejidad computacional de la técnica basada en restricciones concretadas es despreciable, compararemos la complejidad computaciones de los dos algoritmos de obtención de *MUSes*. Asimismo, y ya que ambas estrategias obtienen la misma diagnosis mínima, y no presentan falsos positivos, mostraremos algunos ejemplos obtenidos al aplicar las técnicas al ejemplo en la Figura 1 con distintos datos de entrada.

La Tabla 1 presenta la complejidad computacional de ambos algoritmos de obtención de *MUSes*. Los experimentos han sido realizados usando distintos casos de test con distinto número de restricciones. La tabla muestra la media de tiempo empleado por cada algoritmo y caso de test, así como las iteraciones necesitadas (es decir, el número de subconjuntos de restricciones cuya consistencia ha sido comprobada). Es posible notar que la técnica exhaustiva emplea más tiempo que la basada en vecindad. La razón para esto es que la técnica exhaustiva trata de encontrar los *MUSes* mediante la comprobación de todos los posibles subconjuntos de restricciones, mientras que la técnica basada en vecindad se limita a los subconjuntos formados por vecinos.

A continuación se detallan distintas diagnosis obtenidas utilizando ambas técnicas: la de obtención de *MUSes* y la de restricciones concretadas, que proporcionan los mismos resultados. Para ello se han utilizado distintos casos de test sobre el ejemplo en la Figura 1, partiendo de una asignación de variables satisfactible, y modificando algunos valores para comprobar las desviaciones en el coste:

- Test case 1: establecer un número de horas muy elevado para una de las subfases de la fase de implementación, concretamente la fase de implementación 3. Al realizar la diagnosis, nuestra propuesta determina dos posibles causas de la desviación que se produce: (1) que ha habido una incorrecta planificación, y se han dedicado más horas de las debidas a la implementación 3, o (2) que se han asignado más personas de las necesarias a dicha subfase.

- Test case 2: establecer un número de horas muy elevado para la implementación 3, y un número muy bajo para la implementación 2. La diagnosis indica que el problema es una combinación entre un error en las personas u horas asignadas tanto a la implementación 2 como a la implementación 3, habiendo 4 posibilidades: (1) error en las personas de la implementación 2 y 3, (2) error en las personas de la implementación 2 y las horas de la implementación 3, (3) error en las horas de la implementación 2 y las personas de la 3, o (4) error en las horas de la implementación 2 y la 3.

- Test case 3: establecer un número tanto bajo como elevado de personas para participar en la fase de análisis. La diagnosis determina que, o bien las horas trabajadas en dicha fase, o bien las personas involucradas han sido mal determinadas.

Cuadro 1: Comparación entre los algoritmos de obtención de *MUSes*

#RCN	Técnica exhaustiva		Técnica basada en vecindad	
	miliseg.	iter.	miliseg.	iter.
9	2.588.891	502	12.547	15
9	1.031.110	256	15.437	24
9	1.004.750	256	12.953	15
9	336.000	129	63.844	16
9	114.265	67	48.235	16
9	114.703	67	46.531	12
11	3.354.115	951	32.313	37
11	426.234	131	10.109	16
11	742.188	157	17.438	22
13	2.326.453	514	15.156	21
13	3.004.571	725	21.063	31

7. Conclusiones y trabajos futuros

En este trabajo se propone un nuevo framework para la diagnosis de las desviaciones de costo que se producen en el desarrollo de un proyecto software, estando sus fases representadas mediante un proceso de negocio. Este framework permite diagnosticar una instancia del proceso es función de las reglas de negocio establecidas por la organización, y a partir de los datos establecidos durante la consecución de dicha instancia. Las reglas de negocio son descritas como restricciones que se almacenan en una Base de Datos de Restricciones. Se han definido dos tipos de técnicas basadas en Programación con Restricciones para obtener la mínima diagnosis de una manera eficiente.

En este artículo se han diagnosticado las causas de la desviación de coste en proyectos software. Como trabajo futuro, puede ser posible incorporar el estudio de otros factores que pueden sufrir desviación, como la planificación temporal de los proyectos. De esta forma se conseguiría ayudar a la toma de decisiones en nuevos proyectos con respecto a todos los factores que influyen en el éxito de su desarrollo sin desviaciones. Asimismo, sería posible la ubicación de puntos de control a lo largo del proceso, de forma que se pudiera determinar el cumplimiento del contrato descrito por las reglas de negocio durante la ejecución del proyecto, para poder así corregir las desviaciones antes de su finalización.

Agradecimientos

Este trabajo ha sido parcialmente financiado por la Junta de Andalucía mediante la Consejería de Innovación, Ciencia y Empresa (P08-TIC-04095) y por el Ministerio de Ciencia y Tecnología de España (TIN2009-13714) y el Fondo Europeo de Desarrollo Regional (ERDF/FEDER).

Referencias

[1] Freeman, P., Bagert, D. J., Saiedian, H., Shaw, M., Dupuis, R., y Thompson, J. B., *Software Engineering Body of Knowledge (SWEBOK) In ICSE*, pág. 693-696, 2001.

[2] García de la Banda, M. J., Stuckey, P. J. y Wazny, J., *Finding all minimal unsatisfiable subsets*, PPDP, pág, 32-43, 2003.

[3] Gasca, R. M., Valle, C., Gómez-López, M. T. y Ceballos, R., *NMUS: Structural Analysis for Improving the Derivation of All MUSes in Overconstrained Numeric CSPs*, Current Topics in Artificial Intelligence: 12th Conference of the Spanish Association for Artificial Intelligence, pág. 160-169, 2007.

[4] Ghose, A. y Koliadis, G., *Auditing Business Process Compliance*, ICSOC '07: Proceedings of the 5th international conference on Service-Oriented Computing, 2007.

[5] Gómez-López, M. T., Ceballos, R., Gasca, R. M., y Del Valle, C., *Developing a labelled object-relational constraint database architecture for the projection operator*, Data Knowl. Eng., vol. 68, num. 1, pág. 146-172, 2009.

[6] Guillou, X. L., Cordier, M.-O., Robin, S. y Roze, L., *Monitoring WS-CDL-based choreographies of Web Services*, Proceedings of the 20th International Workshop on Principles of Diagnosis, pág. 43-50, 2009.

[7] Kanellakis, P. C., Kuper, G. M. and Revesz, P. Z., *Constraint Query Languages*, 1992.

[8] Namiri, K., y Stojanovic, N., *A Semantic-based Approach for Compliance Management of Internal Controls in Business Processes*, CAiSE Forum, 2007.

[9] Project Management Institute, *Project Management Institute. Guía de los fundamentos de la Dirección de proyectos*, Project Management Institute, Newtown Square, Pa., 2004.

[10] Sadiq, S. W., Governatori, G., Namiri, K., *Modeling Control Objectives for Business Process Compliance*, BPM, 2007.

[11] Scacchi, W., *Process Models in Software Engineering*, Encyclopedia of Software Engineering. John Wiley and Sons, New York, 2001.

[12] Wallace, L., Keil, M., Rai. A., *Understanding software project risk: a cluster analysis. Information & Management*, 42(1), pag. 115-125, Dic 2004.

[13] Weber, B., Sadiq, S. W., Reichert, M., *Beyond rigidity - dynamic process lifecycle support*, Computer Science - R&D, vol. 23, num. 2, pág. 47-65, 2009.

[14] Weske, M., *Business Process Management: Concepts, Languages, Architectures*, Springer-Verlag, 2007.