

Resolución de acuerdos en Procesos de Negocio para Multiproceso Software usando Programación con Restricciones Distribuidas

Luisa Parody

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

María Teresa Gómez-López

ETS Ingeniería Informática
Univ. de Sevilla
maytegoz@us.es

Rafael Martínez Gasca

ETS Ingeniería Informática
Univ. de Sevilla
gasca@us.es

Diana Borrego

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

Resumen

Un proceso de negocio consiste en un conjunto de actividades que trabajan de manera coordinada para obtener un objetivo común. A veces, la definición del objetivo usando un acuerdo de proceso de negocio clásico no es posible. Cuando la coreografía de procesos no puede definirse como la combinación de tareas usando secuencias, condiciones, puertas 'xor', 'or' y 'split', hay que utilizar otra representación y solución. Este problema hace difícil la toma de decisiones en la gestión de proyectos software. En este artículo proponemos una manera para describir los acuerdos de procesos donde el orden de ejecución de los servicios webs que tienen asociados no está definido. Como caso de estudio utilizamos la distribución de recursos en un entorno de desarrollo de multiproyectos software. En este caso, el proceso tiene que llegar a un acuerdo en función de las reglas de negocio que relaciona a los procesos. Con el fin de obtener este objetivo, utilizamos el modelado y la resolución de Problemas de Satisfacción de Restricciones Distribuidos.

1. Introducción

Una organización dedicada al desarrollo de productos software a medida suele trabajar

en un entorno multiproyecto. La complejidad inherente a la gestión del desarrollo del *software* se multiplica extraordinariamente en presencia de varios proyectos simultáneos que se realizan por una misma organización y que debe repartir los recursos entre ellos. Según el esquema básico del proceso de negocio presentado en [12] para este tipo de organizaciones, la organización planifica la producción de esos proyectos asignándoles recursos para su ejecución en función de su capacidad, de los compromisos adquiridos con los clientes y de la carga global del trabajo. Dicho de otra manera, según los acuerdos de negocio que se establecen para cada uno de los proyectos. Además, la organización dispone de un sistema de control interno a través del cual intenta gestionar la ejecución y modificar las asignaciones para hacer frente a imprevistos que surgen de manera externa o interna a los proyectos. Hay que tener en cuenta que el producto software en general no tiene todas sus características definidas "a priori" sino que se van conformando a lo largo de su propio ciclo de vida y como consecuencia, hace que no sea fácil determinar cuándo se acaba el proceso.

Por otra parte, la organización tiene que establecer unos plazos a cumplir junto con los cliente en cada uno de los proyectos, por lo

que la planificación juega un papel muy importante en el éxito o fracaso de los proyectos. El problema crucial es la administración de los recursos de la organización por los que “compiten” los proyectos concurrentes. Para llegar a una solución tienen que ponerse de acuerdo los diferentes proyectos velando por sus intereses pero cediendo en algunos casos. Cada proyecto tiene establecido un contrato mediante reglas de negocio que son independientes del resto.

La gestión de procesos de negocio ha recibido especial atención desde que permite combinar diferentes tareas para obtener un objetivo común. Un proceso de negocio consiste en un conjunto de actividades que se representan de manera coordinada en un entorno técnico y organizativo [15]. Un acuerdo de proceso de negocio incluye conceptos, métodos y técnicas para soportar el diseño, administración, configuración, construcción y análisis de procesos de negocio. La base de los acuerdos de negocio es la representación explícita de los mismos con sus actividades y las restricciones de ejecución entre ellos. Los aspectos que se analizan en este artículo son aquellos que ocurren cuando el orden de ejecución y las relaciones entre los procesos no pueden describirse con un modelo de proceso de negocio tradicional. Un modelo de proceso de negocio consiste en un conjunto de actividades y restricciones de ejecución entre ellas. Esto ocurre cuando varios procesos tienen que alcanzar un objetivo común y para ello necesitan de un recurso compartido entre todos ellos. En este caso, tienen que llegar a un acuerdo en función de las reglas de negocio que los relacione.

La combinación de procesos puede provocar el uso de servicios con información pública y privada, de manera que las decisiones no se puedan tomar de forma externa ni centralizada a los propios procesos. Hay dos buenas razones para separar los aspectos públicos de los privados en el comportamiento de los procesos de negocio. Uno es que los procesos no quieren revelar todas las decisiones internas que realizan y la gestión de datos al resto de procesos. Y la otra razón es que, incluso cuando no es el caso, separar la información privada y pública de los procesos da la libertad absoluta

para cambiar los aspectos privados de implementación sin afectar al protocolo público del proceso. Esto significa que la información está distribuida en diferentes nodos aunque todos ellos tengan que trabajar juntos.

Teniendo en cuenta todo lo anterior, el trabajo está organizado como sigue: Sección 2 presenta algunas propuestas relevantes existentes en la bibliografía. Sección 3 presenta un ejemplo sobre combinación de servicios. La Sección 3.1 describe el modelado del ejemplo de la sección anterior. La sección 4 explica algunas definiciones y algoritmos relacionados con los problemas de satisfacción de restricciones distribuidos (DisCSP). La sección 5 presenta una propuesta para adaptar las definiciones y algoritmos de DisCSP al problema de los acuerdos de negocios utilizando ejemplo presentado. Sección 6 presenta los resultados obtenidos. Finalmente, se presentan las conclusiones y los trabajos futuros.

2. Trabajos Relacionados

Ante la definición de los acuerdos de negocio, el estándar de Notación para el Modelado de Procesos de Negocio (BPMN) no es suficientemente potente porque, entre otras cosas, existe una importante necesidad de adaptar algunos procesos para que las actividades concurrentes en el proceso respeten las restricciones de coordinación. Esto requiere que las actividades concurrentes coordinen sus comportamientos en respuesta a los eventos de origen externo. En [17] muestra cómo la coordinación de restricciones puede representarse en WS-BPEL, y usa adaptación generalizada y aplicación de modelos de restricción para transformar un proceso BPEL tradicional a uno adaptativo. Una forma de representar las restricciones es utilizando la sintaxis del Lenguaje Semántico de Reglas Web (SWRL) [1] para especificar las restricciones e incrustar las restricciones basadas en SWRL dentro de WS-BPEL. Pero no hay definido un protocolo en BPEL para lograr un acuerdo libre de deadlocks y livelocks. Con el fin de representar la coreografía de procesos en un proceso de negocio, se han desarrollado diferentes lenguajes gráficos. Uno

de los más importantes es el estándar BPMN [15] que ha sido usado en una gran cantidad de procesos de negocio [16].

3. Ejemplo de Proceso para Acuerdos de Negocio

Considerando un Servicio Web basado en procesos para organizar la distribución de personal de una Empresa de Desarrollo Software. Esta empresa se encarga dentro de las fases del ciclo de vida de un proceso software, de la parte de codificación y tendrá que distribuir un conjunto de trabajadores de distinta calidad en tres proyectos distintos que se llevan a cabo de manera concurrente, decidiendo cuántos días trabajarán en cada proyecto. Estos recursos no son estáticos y pueden sufrir cambios originados por ejemplo por baja por enfermedad. También puede ocurrir que por un error muy grave en un proyecto se necesiten refuerzos y por tanto una reestructuración en el resto de proyectos. De acuerdo con la planificación de los proyectos que se está llevando a cabo, el servicio web en respuesta a las contingencias encontradas ayuda proponiendo una reestructuración del personal asignado o simplemente retrasar el tiempo de entrega del proyecto. La ganancia que se obtiene de cada proyecto vendrá derivada de la puntualidad de la entrega del mismo. De formas que una buena división de los trabajadores hará que la suma de las ganancias de los proyectos sea la mayor posible. Las funciones que describen la ganancia de los proyectos son internas a la gestión de cada proyecto, de forma que cada uno de ellos se tendrá que poner de acuerdo para que se cumpla el objetivo global.

3.1. Modelado del Problema

En esta sección se presenta el modelo de un Servicio Web donde el responsable de una empresa define los recursos disponibles y recibe la disposición de personal obteniendo la mayor ganancia. Esto significa que se conoce el número de personas disponibles de cada categoría existente. Si se disponen de tres categorías distintas independientemente de los

proyectos que haya, hay tres variables de entrada dadas por la empresa:

- NumT1: número de trabajadores cuya categoría es de tipo 1.
- NumT2: número de trabajadores cuya categoría es de tipo 2.
- NumT3: número de trabajadores cuya categoría es de tipo 3.

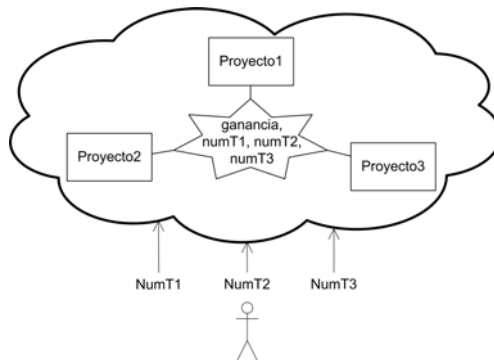


Figura 1: Ejemplo de problema.

3.2. Objetivo de nuestro problema

El objetivo es maximizar la ganancia total de la empresa a partir de los parámetros de entrada. Si se tienen tres proyectos distintos, el cálculo de la ganancia será:

$$\max(\sum \text{Ganancia} = \text{gananciaP1} + \text{gananciaP2} + \text{gananciaP3})$$

La ganancia de cada proyecto depende de los acuerdos de negocio firmado entre el jefe de proyecto y la empresa cliente y los contratos establecidos mediante reglas de negocio por el propio proyecto. Vamos a combinar varios servicios webs correspondientes a cada uno de los proyectos de la empresa para obtener la ganancia total a partir de la ganancia de cada uno de ellos.

3.2.1. Servicio Web Proyecto X

Este servicio web calcula la ganancia del proyecto X a partir del número de trabajadores de tipo 1, tipo 2, tipo 3 para ese proyecto X.

entradas : NumT1, NumT2, NumT3
salida : gananciaPX

Existirá un servicio web por cada proyecto distinto, ya que cada proyecto tiene un contrato distinto para el cálculo de la ganancia.

4. Problema de Satisfacción de Restricciones Distribuido (DisCSP)

Las relaciones entre los distintos Servicios Web pueden ser definidos como un conjunto de restricciones, por esta razón hemos decidido representar la información interna y las relaciones entre ellas como restricciones, donde las variables son los puertos de conexión de los Servicios Web. Si la información fuera toda pública, sería suficiente con construir y resolver un Problema de Satisfacción de Restricciones, pero como la información está distribuida y hay variables privadas será necesario la creación de problemas de satisfacción de restricciones distribuidos.

4.1. CSP

Un Problema de Satisfacción de Restricciones (Constraint Satisfaction Problem - CSP) es un problema de asignación de valores a variables consistente. Un CSP consiste en una terna (X, D, C) donde X es un conjunto de n variables x_1, x_2, \dots, x_n . Para cada variable $x_i \in X$ existe un dominio D_i . El dominio de una variable es el conjunto finito de todos los posibles valores que se le pueden asignar a esa variable, y C es un conjunto finito de restricciones. Una restricción C_k está definida sobre un conjunto de variables mediante el producto cartesiano $D_{k1} \times \dots \times D_{kj}$ los cuales son posibles combinaciones de valores. Una solución a un CSP es una asignación de valores a todas las variables de forma que satisfagan todas las restricciones [2]. Muchos problemas se pueden modelar como un CSP, pero si queremos modelar problemas más realistas, no siempre se puede hacer con un CSP convencional. Es por esta razón que necesitamos modelar el problema con un Problema de Satisfacción de Restricciones Distribuido.

4.2. DisCSP

En [9], Makoto y cia. presentan el Problema de Satisfacción de Restricciones Distribuido (DisCSP) como un formalismo general para tratar con problemas en sistemas multiagentes. Un DisCSP es un CSP donde el conjunto de variables y restricciones del problema está distribuido entre un conjunto de agentes que se encargan de resolver sus propios subproblemas y coordinarse con el resto de agentes para conseguir una solución al problema global [2].

4.2.1. Definiciones

En [8], Hirayama et al. se define un DisCSP como:

- Un conjunto de agentes, $1, 2, \dots, m$.
- Un conjunto de n variables $V = x_1, x_2, \dots, x_n$ donde los valores de las variables pertenecen a un dominio discreto y finito D_1, D_2, \dots, D_n , respectivamente.
- Para cada variable x_j , un agente i es definido tal que x_j pertenece a i . Siendo x_j perteneciente i por la relación (x_j, i)
- Una restricción C_l es conocida por el agente i . El predicado (C_l, i) expresa dicha relación.

Sólo el agente que tiene asignada una variable tienen control sobre su valor y el conocimiento de su dominio. El objetivo de los agentes es elegir los valores de las variables para conseguir la función objetivo global. Asumimos, en general, que un agente conoce sólo las restricciones relevantes a las variables que le pertenecen. Hay que tener en cuenta que algunas restricciones conocidas por un agente incluyen variables de otros agentes, no sólo sus propias variables. A este tipo de restricciones les llamamos restricciones inter-agentes.

Un DisCSP se resuelve cuando todos los agentes satisfacen las siguientes condiciones. Por cada agente i ,

- Una variable x_j tiene un valor d_j tal que su asignación para $\forall x_j$ pertenece (x_j, i) .

- Una restricción C_i es cierta bajo la siguiente asignación $\forall C_i$ conoce (C_i, i) .

En los trabajos relacionados con DisCSP y acuerdos de procesos de negocio se tratan con ejemplos con constantes problemas de deadlock (abrazo mortal) y livelock (bloqueo mutuo), por lo que se han estudiado en ambas áreas. Un deadlock es una situación donde dos o más actividades o procesos están esperando a que otro termine y éste nunca lo hace [14]. El deadlock es un problema común en multiproceso donde algunos procesos comparten una fuente de un tipo específico y mutuamente excluyente, a esto se le conoce como Software lock o Soft lock. Para el tipo de problema propuesto en este paper, la fuente compartida es el conjunto de variables. Un livelock ocurre cuando un proceso A espera la respuesta de otro proceso B y éste proceso B espera la respuesta del proceso A. El resultado es que ninguno de los procesos A y B se completarán, de ese modo, puede ser nula la repetición de estados en el proceso. En nuestro caso viene representado por la instanciación de variables. Los problemas de deadlock y livelock deben estudiarse en el contexto de DisCSP [7], pero las soluciones tienen que analizarse desde un punto de vista de los procesos de negocio.

4.3. Algoritmos para resolver DisCSP

Los algoritmos para CSPs Distribuidos deben encontrar una solución lo más rápido posible. Los agentes asignan valores a las variables, atentos a generar una asignación local consistente que además sea consistente con todas las restricciones entre agentes. Para conseguir este objetivo, los agentes establecen una asignación de valores de sus variables para una consistencia local e intercambian mensajes con otros agentes para comprobar la consistencia de su asignación propuesta para las restricciones con variables pertenecientes a otros agentes [11]. El agente en un DisCSP tiene sólo conocimiento limitado del problema entero, y una de las cosas más importantes para los algoritmos incluye cómo los agentes se comunican con el resto y qué información se transfiere.

Se asumen los siguientes modelos de comunicación [9]:

- Comunicación entre agentes mediante el envío de mensajes. Un agente puede enviar mensajes a otros agentes si y sólo si el agente conoce la dirección del resto de agentes.
- El retraso en la entrega de mensajes es finito, aunque aleatorio. Para la transmisión entre un par de agentes, los mensajes se reciben en el orden en que han sido enviados.

Es posible que puedan combinarse una gran cantidad de Servicios Webs, de modo que hay que analizar cómo escalar las técnicas de resolución concurrentes de los servicios en paralelo [3]. Con el fin de mejorar el tiempo en encontrar la mejor solución, el paper de Redouane Ezzahir et al. [5] proponen un algoritmo para resolver Problemas de Optimización de Restricciones Distribuido (DCOP). El algoritmo se basa en ramificación y acotación con orden dinámico de agentes. Este algoritmo se puede adaptar para encontrar los valores inconsistentes en un espacio cooperativo, ayudando a evitar dinámicamente los sub-problemas inviables y acelerando la búsqueda. Con el fin de modelar la información representada por la información pública y privada, [4] propone un modelo donde las restricciones no son totalmente conocida por ambos agentes. El paper mencionado presenta un nuevo modelo de DisCSP en el que las restricciones se guardan privadas y sólo son parcialmente conocidas por los agentes. La combinación de servicios webs puede implicar obtener la mejor solución o una de las mejores además de provocar construir y resolver un DCOP [13]. Muchos de los estudios en DisCSP apuntan en la definición de algoritmos de búsqueda basados en backtracking distribuido, por lo que vamos a estudiar algunos de ellos.

4.3.1. Backtracking Centralizado

El Backtracking centralizado selecciona un agente líder entre todos los agentes que maneja toda la información sobre variables, sus

dominios y sus restricciones [10]. Si todo el conocimiento del problema se puede centralizar en un único agente, este agente puede resolver sólo el problema usando los algoritmos de satisfacción de restricciones centralizado normal. Pero los algoritmos centralizados en general tiene verdaderos problemas [9] [10]:

- Recolectar toda la información sobre el problema requiere no sólo el coste de comunicación sino el coste de traducir el conocimiento de uno a un formato de intercambio correcto.
- Cada agente tendrá que formular sus limitaciones de antemano. Esto puede ser excesivamente complejo.
- La comunicación puede ser demasiado elevada para seleccionar el líder y que éste recolecte toda la información.
- Pérdida del paralelismo, los otros agentes esperan mientras el líder resuelve el CSP.
- En algunos problemas de aplicación, centralizar toda la información en un agente no es deseado o imposible por razones de seguridad y privacidad.

4.3.2. Backtracking Síncrono

El Backtracking Síncrono asume que todos los agentes están de acuerdo en el orden de instanciación de sus variables (tal que el agente x_1 va primero, luego el agente x_2 y así sucesivamente). Cada agente recibe una solución parcial (las instanciaciones de las variables previas) del agente anterior, e instancia sus variables basándose en las restricciones que conoce. Si encuentra un valor, lo añade a la solución parcial y la pasa al siguiente agente. Si no encuentra una instanciación de sus variables que satisfaga todas las restricciones, envía un mensaje de backtracking al agente anterior haciendo vuelta atrás (backtrack) [9].

Como podemos ver, el orden de las variables es muy importante, se comentará en la Sección 4.3.5.

4.3.3. Backtracking Asíncrono(ABT)

El Backtracking Asíncrono (ABT) se caracteriza por el hecho de que todos los agentes están activos en paralelo y sólo necesitan coordinarse para asegurarse de que sus variables son consistentes con las restricciones.

Desde que ABT fue publicado por [9] en 1998, ha sido una referencia y la base en la construcción de muchos otros algoritmos. Fue formulado para restricciones binarias y nosotros en este artículo vamos a asumir que todas las restricciones son binarias. Cada agente tiene una estructura de datos propia que contiene sus propias variables, constantes que representan el dominio de sus variables, las restricciones propias privadas y las que tienen en común con otras variables y/o agentes, y diferentes estructuras que se usarán para almacenar el estado de la búsqueda. Otras estructuras de datos son *variables* y *nogoods* que serán explicadas con más detalle a continuación. Se utiliza un constructor para indicar que un agente envía un mensaje a otro agente, invocando de esta manera el procedimiento de recepción de mensajes en ese agente receptor. Estas invocaciones se hacen de manera asíncrona, esto significa que no devuelven ningún valor y el proceso de invocación no espera ninguna respuesta para finalizar. Sin embargo, se asume que no se pierde ninguna invocación, todas las invocaciones del mismo agente se manejan en el que se han hecho, y no hay conflictos (Cuando un agente tiene un conflicto, el algoritmo termina, las restricciones que han entrado en conflicto por culpa de los agentes no tienen que satisfacerse en el resultado final).

En ABT el orden de prioridad de las variables y/o agentes está determinado. Cada agente comunica la asignación de valores provisional a sus agentes vecinos via mensajes *ok?*. Cada agente mantiene la asignación de valores concurrente a otros agentes desde su punto de vista, llamado *agent_view*. Un agente cambia su asignación si la asignación de valores concurrente no es consistente con un agente de mayor prioridad. Si no existe valor que sea consistente con los agentes de mayor prioridad, el agente genera una nue-

va restricción, llamada *nogood*, y comunica el *nogood* al agente de mayor prioridad para que dicho agente cambie sus valores haciendo vuelta atrás (backtrack). En [6] [9] [10] prueban la solidez, completitud y la terminación de los algoritmos ABT.

La principal limitación tanto del ABT como del Backtracking Síncrono es que el orden de los variables y/o agentes está determinado estáticamente. Si el valor de selección del agente prioritario es malo, los agentes menos prioritarios necesitan realizar una búsqueda exhaustiva para revisar y corregir esa mala decisión. En la Sección 4.3.5 vamos a hablar sobre esto.

4.3.4. Asynchronous Weak-Commitment (AWC)

El Asynchronous Weak-Commitment (AWC) [10] es una variante del ABT. Introduce la heurística de mínimos conflictos para reducir el riesgo de hacer malas decisiones. Los agentes prioritarios se ajustan dinámicamente cuando ocurre la vuelta atrás (backtrack), el agente que inicia la vuelta atrás se convierte en el agente más prioritario. Se construye una solución parcial consistente para un subconjunto de variables, y esa solución parcial se extiende añadiendo el resto de variables una por una hasta que se encuentra una solución completa [18]. El mayor inconveniente es que AWC es completo sólo si todos los *nogoods* se almacenan, dando lugar a un aumento exponencial en los requisitos de almacenamiento.

4.3.5. Heurísticas para la ordenación de variables

El Backtracking síncrono explora un árbol de búsqueda con un orden de variables fijo que asume ser x_1, x_2, \dots, x_n sin perder generalidad. El orden fijado es conocido por todos los agentes, y establece *prioridades* en el que una variable x_i tiene prioridad sobre otra variable x_j cuando $i < j$. Notar que esto no implica que un agente tenga conocimiento del problema entero: el orden puede establecerse por ejemplo asignando a cada agente un único número y dejando la ordenación correspondiente a ese número.

El orden de las variables se usa para decidir la dirección en cada restricción: el agente controlador de la primera variable en el orden se llama agente *value – sending* (enviador de mensaje), y el otro se llama agente *constraint – evaluating* (evaluador de la restricción).

Por otro lado, ABT asume un orden de prioridad estático entre todos los agentes. Los agentes más prioritarios llevan a cabo una asignación y la envían vía mensajes a los agentes menos prioritarios. ABT asume que los agentes menos prioritarios que están involucrados en la restricción pueden comprobar todas las restricciones inter-agentes, de ese modo, el agente menos prioritario puede manejar toda la restricción. En ambos algoritmos, hay diferentes formas de ordenar las variables para establecer la prioridad:

1. La variable con mayor prioridad será aquella que participe en más restricciones. Además, hay diferentes posibilidades para establecer la siguiente variable más prioritario:
 - a) La siguiente variable más prioritaria será aquella que participe en más restricciones sin tener en cuenta a qué agente pertenece.
 - b) La siguiente variable más prioritaria será aquella que participe en más restricciones y que pertenezca al mismo agente de la primera variable.
2. Las variables más prioritarias pertenecerán al agente más rápido. El agente más rápido es aquel cuyo Servicio Web devuelve el resultado más rápido. Las siguientes variables más prioritarias serán aquellas que pertenezcan al siguiente agente más rápido, y así sucesivamente.
3. La prioridad se establecerá como mezcla de los apartados anteriores.
4. El orden de las variables será establecido por el diseñador.

5. DisCSP aplicado al problema de gestión de proyectos software

En el caso particular de nuestro ejemplo, habrá tres proyectos que trabajan en paralelo y necesitan la organización del personal. Aunque estos proyectos pertenecen a la misma empresa, no tienen por qué tener los mismos acuerdos con el cliente y contratos internos, y por razones de seguridad y privacidad, sólo comparten entre ellos la información necesaria para alcanzar el objetivo común. Cada proyecto se corresponderá a un agente a los que llamaremos Agente Proyecto 1, Agente Proyecto 2 y Agente Proyecto 3. Estos tres agentes se tendrán que poner de acuerdo para maximizar la ganancia de la empresa intentando terminar su proyecto en los plazos establecidos y con el personal disponible.

Todos los agentes tendrán que tener en cuenta las siguientes restricciones comunes para instanciar sus variables:

(C1)

$$\begin{aligned} numT1 &= numT1P1 + numT1P2 + numT1P3 \\ numT2 &= numT2P1 + numT2P2 + numT2P3 \\ numT3 &= numT3P1 + numT3P2 \end{aligned}$$

Se considera que los trabajadores sólo pueden estar asignados a un único proyecto. Si por ejemplo sólo hay disponibles 5 trabajadores de tipo 1, entre los tres proyectos no puede haber más de 5 trabajadores de tipo 1.

5.1. Agente Proyecto 1

Este agente se encarga del cálculo de la ganancia cumpliendo el contrato del Proyecto 1. Se puede modelar como un CSP particular tal como se muestra en la Figura 2.

Las variables son:

- *gananciaP1* la ganancia total del Proyecto 1.
- *diasEstP1* número de días estimados para realizar el Proyecto 1.
- *numT1P1*, *numT2P1* y *numT3P1* el número de trabajadores que habrá de tipo 1, 2 y 3 en el Proyecto 1 respectivamente.

<p>Variabes:</p> <p><i>gananciaP1</i> : {0..550000} <i>numT1P1</i> : {0..numT1} <i>numT2P1</i> : {0..numT2} <i>numT3P1</i> : {0..numT3} <i>numDiasTrabT1P1</i>: {0..365} <i>numDiasTrabT2P1</i>: {0..365} <i>numDiasTrabT3P1</i>: {0..365} <i>diasEstimadosP1</i>: {5..5} <i>diasTrabajadosP1</i>: {0..365} <i>lcP1</i> : {57000..57000} <i>puntualidadP1</i>: {0..100} ...</p> <p>Restricciones:</p> <p><i>gananciaP1</i> == 5500 * <i>puntualidadP1</i> <i>lcP1</i> == <i>numT1P1</i> * <i>numDiasTrabT1P1</i> * 1000 + <i>numT2P1</i> * <i>numDiasTrabT2P1</i> * 500 + <i>numT3P1</i> * <i>numDiasTrabT3P1</i> * 200 <i>diasTrabajadosP1</i> == max(<i>numDiasTrabT1P1</i>, <i>numDiasTrabT2P1</i>, <i>numDiasTrabT3P1</i>) <i>puntualidadP1</i> == min(100, (100 * <i>diasEstimadosP1</i>) / <i>diasTrabajadosP1</i>) ...</p>
--

Figura 2: Variables y Restricciones del Agente Proyecto 1.

- *numDiasTrabT1P1*, *numDiasTrabT2P1* y *numDiasTrabT3P1* son el número de días trabajados por los trabajadores de tipo 1, 2 y 3 en el Proyecto 1 respectivamente.
- *diasTrabajadosP1* duración total del Proyecto 1.
- *lcP1* número de líneas de código que hay que realizar en el Proyecto 1.
- *puntualidadP1* puntualidad en la entrega del Proyecto 1, valdrá 100 si se entrega antes de tiempo o en el tiempo estimado y menor que 100 si se retrasa.

Las restricciones establecen que la ganancia total del Proyecto 1 dependerá de la puntualidad. Si se forma puntualmente se obtendrá una ganancia de 550000 euros, que es lo que establece el contrato. El retraso en la entrega del proyecto trae consigo pérdidas que se traducen en la disminución de dicha ganancia. La puntualidad viene dada por el cociente entre el tiempo estimado y la duración total del

proyecto. La duración total del proyecto será el máximo entre los días trabajados por los trabajadores de las distintas categorías. Por su parte, el Proyecto 1 establece que los trabajadores de tipo 1 son capaces de realizar 1000 líneas de código al día, mientras que los de tipo 2 y 3 son capaces de realizar 500 y 200 líneas de código al día respectivamente. El cálculo total de líneas de código del Proyecto 1 depende del número de trabajadores de cada categoría por el número de días que trabajan y el número de líneas de código que son capaces de realizar por día. Cuando se llega como mínimo al número de líneas de código establecido el proyecto se considera terminado. Se estima que el proyecto va a tener una duración de 5 días.

5.2. Agente Proyecto 2

Este agente se encarga del cálculo de la ganancia cumpliendo el contrato del Proyecto 2. Al igual que ocurría con el Agente Proyecto 1, se puede modelar como un CSP particular tal como se muestra en la Figura 3.

<p>Variables: <i>diasEstimadosP2</i>: {7..7} <i>lcP2</i>: {37000..37000} ...</p> <p>Restricciones: $gananciaP2 == 1000 * puntualidadP2$ $lcP2 == numT1P2 * numDiasTrabT1P2 * 1000 +$ $numT2P2 * numDiasTrabT2P2 * 700 +$ $numT3P2 * numDiasTrabT3P2 * 500$ $diasTrabajadosP2 == max(numDiasTrabT1P2,$ $numDiasTrabT2P2, numDiasTrabT3P2)$ $puntualidadP2 == min(100,$ $(100*(diasEstimadosP2+5))/diasTrabajadosP2)$...</p>

Figura 3: Variables y Restricciones del Agente Proyecto 2.

Las variables son iguales que las del Agente Proyecto 1, cambiando en el nombre *P1* (Proyecto 1) por *P2* (Proyecto 2). En cuanto a las restricciones, si en el Proyecto 2 se es puntual se obtiene una ganancia menor, de 100000 *euros*, sin embargo se establece que se

es 100 % puntual si el proyecto dura el tiempo estimado más un margen de 7 días siendo el tiempo estimado de 5 días. Además, el número total de líneas de código a realizar es de 37000 y se establece que los trabajadores de tipo 2 y 3 son capaces de realizar 700 y 500 líneas de código al día respectivamente.

5.3. Agente Proyecto 3

Este agente se encarga del cálculo de la ganancia cumpliendo el contrato del Proyecto 3. Al igual que ocurría con el Agente Proyecto 1 y Agente Proyecto 2, se puede modelar como un CSP particular tal como se muestra en la Figura 4.

<p>Variables: <i>diasEstimadosP3</i>: {10..10} <i>lcP3</i>: {20000..20000} ...</p> <p>Restricciones: $gananciaP3 == 3500 * puntualidadP3$ $lcP3 == numT1P3 * numDiasTrabT1P3 * 1000 +$ $numT2P3 * numDiasTrabT2P3 * 500$ $diasTrabajadosP3 == max(numDiasTrabT1P3,$ $numDiasTrabT2P3)$ $puntualidadP3 == min(100,$ $(100*(diasEstimadosP3+diasEstimadosP3/4))/$ $diasTrabajadosP3)$...</p>

Figura 4: Variables y Restricciones del Agente Proyecto 3.

Las variables son iguales que las del Agente Proyecto 1 y Agente Proyecto 2 con la salvedad de que el contrato de este Proyecto considera que no puede haber ningún trabajador de tipo 3, por lo que todas las variables relacionadas con el tipo 3 desaparecen. En cuanto a las restricciones, si en el Proyecto 3 se es puntual se obtiene una ganancia de 350000 *euros*, además establecen que se es 100 % puntual si el proyecto dura el tiempo estimado más un margen del 25 % de los días estimados siendo el tiempo estimado de 10 días. El número total de líneas de código a realizar es de 20000 y se establece que los trabajadores de tipo 1 y 2 son capaces de realizar 1500 y 700 líneas de código al día respectivamente.

5.4. Backtracking aplicado al problema

En este trabajo se propone resolver los problemas de acuerdo entre tareas para maximizar objetivo basándonos en los algoritmos propuestos en la Sección 4.3 con una necesaria adaptación a procesos de Negocio.

El primer paso es definir el algoritmo de Backtracking para modelar el proceso de negocio (ver el Algoritmo 1). La clave está en la línea 8, donde el algoritmo llama a los diferentes agentes para calcular la ganancia. Este algoritmo lo podríamos clasificar como una mezcla entre Backtracking centralizado y Asíncrono. Centralizado porque el Proceso de Negocio tiene conocimiento de todas las variables públicas y sus dominios y Asíncrono porque todos los agentes están activos en paralelo y comunican sus consistencias o inconsistencias de manera asíncrona.

```

1:  $V :=$  Lista de variables ordenadas  $\{v_1, \dots, v_k\}$ 
   y  $\{v_1 = \{val_{11} \dots val_{1n}\}, \dots, v_k = \{val_{k1} \dots val_{km}\}\}$ 
   sus posibles valores.
2:  $STAGE :=$  par (Clave, Valor) donde,
    $Clave_i = \{v_i \in V\}$  y  $Valor_i = \{val_{ix} \in v_i\}$ .
3:  $SOL :=$  solución que contiene la lista de
   valores de las variables y la ganancia
   asociada.
4:  $SOL\_OP :=$  solución óptima.
5: Mientras exista  $v \in V$  hacer
6: Mientras exista  $val \in v$  hacer
7:    $STAGE \leftarrow (v, val)$ 
8:    $GAN :=$  Calcular Ganancia
9:   Si  $GAN < SOL\_OP.GAN$  entonces
10:     $SOL \leftarrow (v, val)$  Y  $GAN$ 
11:    Si  $\forall v \exists val$  en  $SOL$  entonces
12:      Si  $SOL.GAN < SOL\_OP.GAN$ 
      entonces
13:         $SOL\_OP \leftarrow SOL$ 
14:      finSi
15:    sino
16:      //Salir del Mientras y continuar
      // con la siguiente variable.
17:    finSi
18:  finSi
19: //Hacer backtrack si no hay más valores
  //para la variable  $v$ 
20: finMientras
21: //Terminar si no hay más valores para
  //las variables
22: finMientras

```

Algoritmo 1: Algoritmo de Backtracking Iterativo.

Tal como se presenta en la Sección 4.3.5 y en la línea 1 del Algoritmo de Backtracking Iterativo, el orden de las variables es muy importante. El segundo paso, por lo tanto, es estudiar las variables comunes y las restricciones para establecer las diferentes prioridades.

- Las variables comunes son: $numT1$, $numT2$, $numT3$, $ganancia$.
- Las variables públicas pertenecientes a los agentes son: $numT1P1$, $numT2P1$, $numT3P1$, $numT1P2$, $numT2P2$, $numT3P2$, $numT1P3$, $numT2P3$, $gananciaP1$, $gananciaP2$, $gananciaP3$.
- Las restricciones comunes son la función objetivo y la restricción C1.

Teniendo en cuenta los casos estudiados en la Sección 4.3.5 y el problema particular, tenemos que todas las variables participan en el mismo número de restricciones y no hay diferencias significativas en la rapidez de los diferentes agentes. En este caso, para establecer el orden de prioridades se va a tener en cuenta la función objetivo del problema: maximizar la ganancia. Las variables más prioritarias serán aquellas que pertenezcan al proyecto que da mayor ganancia. Este criterio favorece el encontrar una solución buena antes y realizar cotas más eficientes. El orden de agentes sería: Agente Proyecto 1, Agente Proyecto 3 y Agente Proyecto 2.

El tercer paso es trasladar el Algoritmo de Backtracking Iterativo a Proceso de Negocio. El resultado se muestra en la Figura 5.

La actividad más importante en el Proceso de Negocio es “Calcular GANANCIA”, tal como ocurría en el algoritmo. En esta actividad, Figura 6, hay en paralelo tres flujos pertenecientes a cada agente. Cada agente tiene dos actividades:

- Resolver las variables del Proyecto: usa las restricciones para asignar valores consistentes a las variables.
- Establecer la ganancia: llama al Servicio Web que calcula la ganancia a partir de los valores asignados a la variable en la actividad anterior.

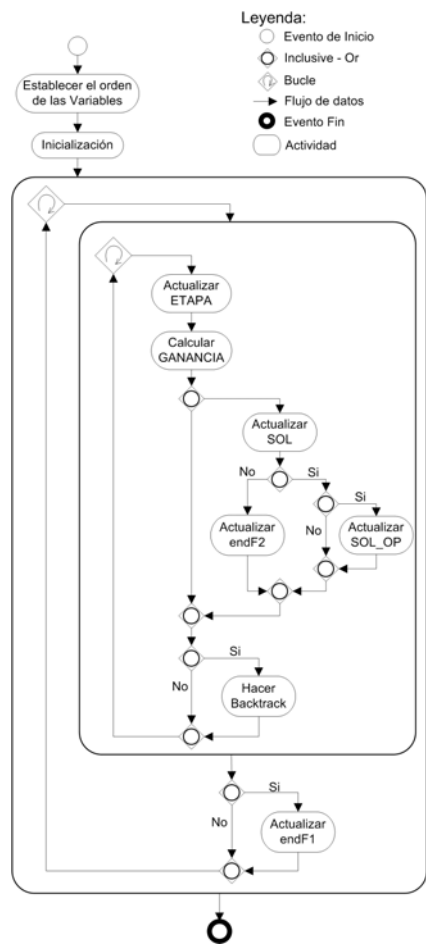


Figura 5: Proceso de Negocio de Backtracking.

5.5. Posibles mejoras

Uno de los principales problemas en la búsqueda de soluciones es la amplitud del espacio de búsqueda. La búsqueda podría finalizar cuando se encuentra una solución en la que todos los Agentes terminan sus proyectos dentro de plazo, independientemente de la asignación de personal. Además, se podría establecer una cota para no tener que llamar a los Agentes y calcular la ganancia si sabemos que no vamos a obtener un resultado mejor. Esta cota puede obtenerse de la memoria "post-mortem"[12] de

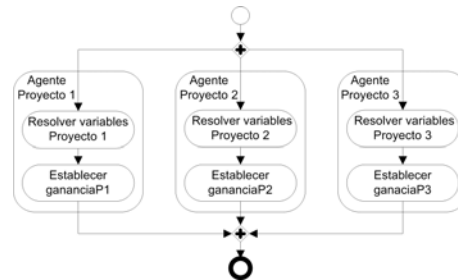


Figura 6: Proceso de Negocio para el cálculo de la Ganancia.

proyectos anteriores que hace posible analizar determinadas incidencias que han podido afectar al perfil de desarrollo de dichos proyectos. También puede establecerse a partir del espacio de búsqueda ya realizado.

6. Resultados

Se establecen como parámetros de entrada que el número de trabajadores total de tipo 1, 2 y 3 para la empresa de software descrita durante todo el trabajo es de 5, 10 y 6 respectivamente. El resultado que se obtiene es de una ganancia total de 840500 euros. La distribución resultante del personal para cada proyecto se muestra en el Cuadro 1.

Cuadro 1: Resultados

Variables	P1	P2	P3
num. de T1	4	0	1
num. de T2	8	1	1
num. de T3	1	5	-

7. Conclusiones y Trabajos Futuros

Este trabajo presenta el proceso de negocio resultante de adaptar el algoritmo de Backtracking para CSP Distribuidos a procesos de negocios. Esta idea surge de la necesidad de coordinar diferentes actividades pertenecientes a un proceso de negocio que trabajan en paralelo y que comparten uno o más recursos. Gracias a

esta adaptación las distintas actividades consiguen coordinarse para obtener un objetivo común, en nuestro caso particular, la ganancia máxima.

Como trabajos futuros proponemos modelar e implementar la solución que adapta el resto de algoritmos que existen para DisCSP para procesos de negocios. El modelado puede ampliarse para realizar una descripción más realista del problema, la ganancia depende de más factores además de la puntualidad en la entrega de un proyecto. Identificar y resolver posibles problemas sobre restringidos que ocurren cuando no existe una solución porque la evaluación de las variables nunca satisfacen todas las restricciones o los agentes no se ponen de acuerdo sobre los valores de las variables.

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] *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. W3C, 2004.
- [2] M. Abril López, F. Barber Sanchís, and M. A. Salido Gregorio. *Particionamiento y Resolución Distribuida Multivariable de Problemas de Satisfacción de Restricciones*. Valencia, 2007.
- [3] L. Bordeaux, Y. Hamadi, and H. Samulowitz. Experiments with massively parallel constraint solving. In *IJCAI*, pages 443–448, 2009.
- [4] I. Brito, A. Meisels, P. Meseguer, and R. Zivan. Distributed constraint satisfaction with partially known constraints. *Constraints*, 14(2):199–234, 2009.
- [5] R. Ezzahir, C. Bessiere, I. Benelallam, H. Bouyakhf, and M. Belaïssaoui. Dynamic backtracking for distributed constraint optimization. In *ECAI*, pages 901–902, 2008.
- [6] B. Faltings. Chapter 20 distributed constraint programming. In P. v. B. Francesca Rossi and T. Walsh, editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 699 – 729. Elsevier, 2006.
- [7] K. Hirayama and J. Toyoda. Forming coalitions for breaking deadlocks. In *ICMAS*, pages 155–162, 1995.
- [8] K. Hirayama and M. Yokoo. Distributed partial constraint satisfaction problem. 1330:222–236, 1997.
- [9] E. H. D. Makoto Yokoo, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Transactions on knowledge and data engineering*, 10(5):673–685, 1998.
- [10] K. H. Makoto Yokoo. Algorithms for distributed constraint satisfaction: A review. In *AAAI*, volume 3, pages 198–212, 2000.
- [11] P. Meseguer, N. Bouhmala, T. Bouzoubaa, M. Irgens, and M. Sánchez. Current approaches for solving over-constrained problems. *Constraints*, 8(1):9–39, 2003.
- [12] J. Navascués Fernández-Victorio and I. Ramos Román. *Un modelo para la simulación híbrida de la producción de software a medida en un entorno multiproyecto*. Sevilla, 2008.
- [13] M.-C. Silaghi and M. Yokoo. Adopt-ing: unifying asynchronous distributed optimization with asynchronous backtracking. *Autonomous Agents and Multi-Agent Systems*, 19(2):89–123, 2009.
- [14] C. Stahl and K. Wolf. Covering places and transitions in open nets. In *BPM '08: Proceedings of the 6th International Conference on Business Process Management*, pages 116–131, Berlin, Heidelberg, 2008. Springer-Verlag.
- [15] M. Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer, 2007.
- [16] C. Wolter and A. Schaad. Modeling of task-based authorization constraints in bpmn. In *BPM*, pages 64–79, 2007.
- [17] Y. Wu and P. Doshi. Making bpel flexible - adapting in the context of coordination constraints using ws-bpel. pages 423–430, 2008.
- [18] M. Yokoo. Weak-commitment search for solving constraint satisfaction problems. In *AAAI*, pages 313–318, 1994.