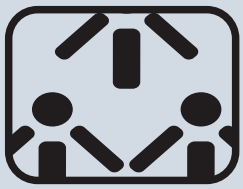
Una técnica basada en componentes para la resolución distribuida de problemas



LUIS IRIBARNE
Doctor Ingeniero en Informática y Docente Titular de la Escuela Politécnica Superior (EPS) de la Universidad de Almería.
Departamento de Lenguajes y Computación
e-mail: luis.iribarne@ual.es
04120 Almería, España

RESUMEN.

En el campo de las ingenierías, en general, es importante disponer de pautas y procesos que faciliten las actividades en la resolución de problemas complejos de la ingeniería, principalmente en los de simulación, donde la carga de cálculo es considerablemente alta y las partes en las que se descompone el problema están débilmente acopladas. En este artículo se propone un marco de trabajo de un sistema basado en componentes y centrado en agentes autónomos que ofrece soporte para la resolución distribuida de problemas. La técnica parte de un plan de trabajo y de una colección de componentes encargados de generar, optimizar y distribuir las tareas del plan, y de una colección de agentes encargados de ejecutarlas. Asimismo, se propone una arquitectura de componentes para el sistema de resolución y un algoritmo de optimización de planes de tareas.

PALABRAS CLAVES
Sistema basado en componentes, agentes autónomos, resolución distribuida de problemas.

1. INTRODUCCIÓN

La informática distribuida ha evolucionado de forma importante en estos años en todas sus áreas de intervención (hardware, software y middleware), lo que ha permitido la aparición de nuevos escenarios donde consolidar y ampliar técnicas, metodologías y entornos distribuidos. Un ejemplo de esto es el fuerte crecimiento de los sistemas multiagente (SMA) y de componentes dentro de la informática distribuida, debido, en gran medida, al explosivo avance tecnológico de las comunicaciones, de Internet, y de especificaciones de distribución como CCM [8], pero también debido a otros aspectos, como a la modelización de agentes autónomos y su organización dentro de un entorno de agentes y componentes, la formalización de comportamiento y la estructura de los agentes y componentes, la identificación y modelización de planes de ejecución y aprendizaje de un sistema multiagente, y al estudio de lenguajes para la programación y experimentación con agentes y componentes.

La inteligencia artificial distribuida (IAD) estudia e intenta construir conjuntos de entidades autónomas e inteligentes que cooperan para desarrollar un trabajo y se comunican por medio de mecanismos basados en el envío y recepción de mensajes. Esta definición va más allá aún cuando numerosos

autores justifican la IAD como aquel campo de la informática que engloba áreas como la inteligencia artificial (IA), encargada de construir entidades inteligentes, y áreas como los sistemas distribuidos, que estudian las propiedades de un conjunto de procesadores que cooperan entre sí comunicándose por mensajes. Por tanto, todas estas definiciones conducen a poder afirmar que la IAD estudia los modelos y comportamientos entre agentes que cooperan para resolver un problema o desarrollo de una tarea, o que la IAD es el área de la informática que estudia tres campos [7]:

(a) Los sistemas multi-agente (SMA): rama de la IAD que estudia el comportamiento de agentes inteligentes que resuelven un problema de manera cooperativa.
(b) La resolución distribuida de problemas (RDP): rama de la IAD que estudia la forma de dividir un problema y asignar cada parte del mismo a un conjunto de entidades independientes para lograr una solución.
(c) La inteligencia artificial en paralelo (IAP): rama de la IAD que se centra en el desarrollo de lenguajes y algoritmos paralelos para sistemas concurrentes en la IAD.

Los sistemas de resolución distribuida de problemas (SRDP) se caracterizaban, entre otras cosas, por ser unos sistemas de actuación concurrente entre diferentes nodos de red, coordinados para resolver un problema concreto, y controlados, como norma general, por un gestor centralizado que organizaba la secuencia de ejecución. Ejemplo de esto era la arquitectura de pizarra del sistema Hersay [4]. De hecho, el proceso de 'razonamiento' de los distintos componentes de un RDP se puede desglosar en cuatro etapas [9]:

(a) Descomposición. Descomposición progresiva del problema general en subproblemas o tareas menos complejas hasta llegar a obtener tareas básicas o primitivas en las que no se pueden dividir en otras.
(b) Asignación. Asignación de las tareas y recursos del sistema a entidades independientes (agentes) que funcionan en nodos diferentes de la red. En este caso habrá que determinar qué tareas deberá resolver un determinado agente y a qué recursos podrá acceder.
(c)Resolución. Resolución de las tareas. Cada agente del sistema intentará resolver las tareas encomendadas según sus posibilidades.
(d) Integración. Integración de soluciones. Por regla general, el gestor del sistema recogerá las distintas soluciones ofrecidas por los agentes e intentará componerlas, para dar con la solución global del problema.

Los autores, fieles a esta definición (o definiciones parecidas), reconocen la existencia de entidades independientes que funcionan de forma autónoma, que incluso coinciden en llamarlas como "agentes", y que son los que se encargan de resolver cada una de las tareas básicas en las que se descompone el problema global. No obstante, según estos autores, los agentes de un SRDP no tienen validez dentro de un sistema multiagente (SMA) ya que éstos, en su conjunto, se ven sometidos a continuos cambios locales, mientras que en los primeros (SRDP) los mecanismos de funcionamiento global son siempre los mismos [10]. Parecería posible, no obstante, pensar en las ventajas que ofrecen tanto los SRDP como los SMA para que estos sean integrados y generar así pautas de resolución.

Si bien es cierto que en un SRDP las pautas de funcionamiento están de antemano predefinidas, también es cierto que al considerar un sistema prácticamente descentralizado (como es el caso de los SMA), donde el plan de ejecución de tareas no esté controlado por un gestor, sino que el gestor sea el propio sistema, hace pensar si ambas definiciones convergen. De hecho, todo SMA se crea y funciona para resolver un determinado problema particular o colectivo, dependiendo del escenario donde se desarrolle el sistema.

Estas facilidades de integración entre sistemas son posibles mediante el uso de pautas de desarrollo de los sistemas basados en componentes (SBC) [1], las cuales aglutinan técnicas orientadas a objetos, en descomposición de problemas y técnicas para el desarrollo de software abierto [2].

En este artículo, se propone un marco de trabajo para la resolución distribuida de problemas basado en el comportamiento de un sistema de agentes autónomo y en las pautas de desarrollo de los sistemas basados en componentes. La técnica propuesta está pensada para la resolución de problemas complejos de la ingeniería, donde la carga de cálculo es considerablemente alta y las partes en las que se descompone el problema están débilmente acopladas, ii.e., se permita cierto grado de paralelismo para la resolución del mismo. Además, las entidades autónomas (agentes) del sistema propuesto coexisten dentro de un marco distribuido cumpliendo parte de las propiedades de un SMA [3]:

- La heterogeneidad: cada entidad puede estar diseñada a partir de representaciones internas, arquitecturas y lenguajes diferentes.
- Homogeneidad de intereses: cada entidad del sistema es programada para colaborar en beneficio propio del sistema, ii.e., resolver el problema.

Este trabajo se organiza en seis secciones, siendo la primera de ellas la presente introducción. La Sección 2 describe la técnica de resolución de problemas basada en características de los sistemas de agentes y componentes, y una descripción de la arquitectura del sistema. La Sección 3 describe el proceso de interacción entre los componentes principales del sistema. La Sección 4 presenta un algoritmo de optimización de planes de trabajo. La Sección 5 presenta unas conclusiones y trabajo futuro. El artículo finaliza con unas referencias.

2. DESCRIPCIÓN DE LA TÉCNICA

Como se ha adelantado, la técnica se centra en un sistema para la resolución distribuida de problemas basado en un sistema de agentes autónomo y de componentes. En este sentido, se puede pensar en problemas concretos definidos por el usuario indicando los parámetros mínimos para hacer la descomposición jerárquica del problema en un conjunto de tareas que se distribuirán entre los agentes del sistema.

2.1 EL PROCESO

En el sistema cohabitan N ambientes de cómputo

(computadoras o procesadores), cada una de ellas con un agente autónomo en ejecución en primer plano, a la espera de recibir un plan de trabajo al cual responder. Uno de los ambientes de cómputo dispone de un programa con la interfaz de usuario desde donde se define el problema a resolver, problema que será aceptado por otro agente o planificador, encargado de generar un plan de ejecución en un formato estándar para el resto de los agentes en espera. Este planificador podrá residir en el mismo ambiente donde se ejecuta la interfaz de usuario, o en otro distinto. Representar un plan de trabajo supone la creación de algún lenguaje para la planificación de problemas, junto con un compilador que analice las dependencias entre las tareas del plan definido a partir de dicho lenguaje. Posteriormente, un algoritmo de filtrado detecta las incongruencias entre las dependencias de las tareas, un error que puede ser introducido por el usuario es por la realización poco detalla del análisis del problema. Aunque este apartado podría ir incorporado en el compilador del lenguaje, es preferible mantener la autonomía de las acciones.

Una vez definido y depurado el plan de ejecución, habrá que decidir cómo distribuir el control de las tareas reflejadas en el plan general de ejecución entre los agentes que esperan en cada uno de los ambientes de cómputo del sistema. La distribución es centralizada, un agente autónomo (organizador) recoge el plan de ejecución generado por el planificador y estudia la forma de distribuir todas las tareas entre los distintos agentes del sistema. El organizador sólo deberá conocer el número de ambientes de cómputo (agentes) dispuestos a resolver el problema global, y el plan de tareas generado por el planificador. Un algoritmo de distribución detecta, a partir del plan de ejecución, el nivel de "paralelización" máxima de dicho plan y efectúa la asignación directa de tareas a cada ambiente de cómputo. Para el caso de la distribución descentralizada, no existirá un gestor que determine la distribución de las tareas del plan; los propios agentes del sistema deciden qué tareas podrán realizar, i.e., cuando el planificador genera el plan de ejecución, éste (el plan) se ofrece al conjunto de agentes del sistema, y son éstos los que se ponen de acuerdo para organizar y coordinar dicho plan. El presente trabajo se centra sólo en el modelo de distribución centralizado.

2.2 ARQUITECTURA GENERAL

El marco de trabajo desarrollado está enfocado a la resolución de problemas complejos con una elevada carga de cálculo y un débil acoplamiento entre sus tareas, como lo son todos aquellos problemas de simulación en la ingeniería, p.e., el modelado de invernaderos, aeronaves, y sistemas robotizados, entre otros. La mayor parte de estos problemas basan su modelización en el cálculo por elementos finitos, donde gran parte de las tareas se aplican de forma iterativa sobre diferentes partes del objeto o caso a simular, sin tener dependencias las unas sobre las otras.

La arquitectura general del sistema propuesto, que se muestra en la Figura 1, está compuesta por los siguientes componentes: (1) una colección de k agentes que esperan un plan de ejecución, (2) un generador de planes, GP; (3) un optimizador de planes, OP; (4) un distribuidor de planes, DP; y (5) un espacio de trabajo, ET. La arquitectura está diseñada de tal forma que cada componente de la misma puede funcionar indistintamente de forma autónoma en uno o más ambientes de cómputo; como requisito, que todos los componentes deban conocer el recurso compartido: el espacio de trabajo (ET). Esto es, si se decide distribuir todos los componentes del sistema en ambientes de cómputo distintos, entonces necesitaríamos k+4 ambientes (véase de nuevo la Figura 1), aunque esta situación no suele ser la normal.

El flujo de ejecución entre los componentes del sistema es el que se describe a continuación, aunque más adelante se hará una descripción detallada de cada uno de ellos por separado.

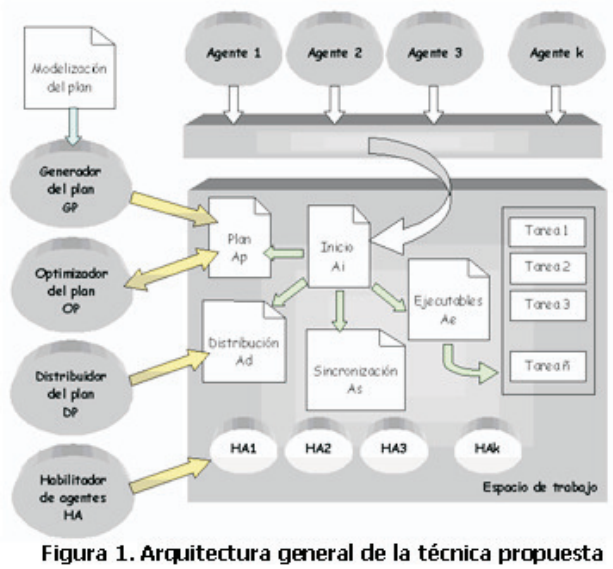


Figura 1. Arquitectura general de la técnica propuesta

Por un lado se activan los k agentes que actualizan sus perfiles de comportamiento desde un componente de inicialización (Ai). Una vez activados los perfiles, los agentes permanecen a la espera de que se genere un "testigo" de habilitación para cada uno de ellos (Ha_i). Este testigo de habilitación no se genera hasta que esté completo el plan de ejecución y se conozca la forma de distribuir las tareas en el sistema. Por otro lado, el generador de planes (GP) recoge el código de modelización del plan, utilizando alguna técnica de representación de planes, y lo transforma a un formato que reconoce el resto de las componentes del sistema. El optimizador del plan (OP) analiza el plan general y elimina incongruencias de estado entre tareas, volviendo a regenerar el plan definitivo con el mismo formato anterior. Seguidamente, el distribuidor del plan (DP) detecta la forma de asignar partes del plan a cada uno de los agentes que permanecen en espera, generando un plan de distribución (Ad). Por último, el habilitador de agentes (HA) genera k testigos de habilitación, uno para cada agente en espera. Una vez creados, los agentes detectan su presencia en el espacio de trabajo y comienzan a funcionar. En primer lugar, todos ellos leen el plan de distribución (Ad) para comprobar qué parte del plan les corresponde. Una vez organizados, obtienen su parte del plan de trabajo, y comienzan a trabajar. Los agentes se comunican y coordinan a través del componente de sincronización (As). Este componente se basa en el mecanismo de pizarra de Hersay [4], donde los agentes van anotando el código las tareas que han ido finalizando. Por otro lado, el componente de ejecución (Ae) determina, a partir del código de la tarea, el componente que ejecutará la tarea específica. Cuando un agente finaliza su parte del plan, eliminará el testigo de habilitación generado por el habilitador de agentes (HA), y permanece de nuevo en espera, solicitando un nuevo plan.

2.3 EL PLAN

El plan de trabajo (Ap) es una de las partes más importantes del sistema. En él se indican las dependencias entre las tareas. Para detectar estas dependencias es necesario utilizar alguna técnica de modelización de planes. Para desarrollar los planes de trabajo de los experimentos de validación del sistema propuesto se ha utilizado un esquema basado en un grafo de dependencias junto con una representación matricial del mismo, representación utilizada por el componente optimizador del plan, que posteriormente comentaremos.

Para describir mejor el funcionamiento del sistema propuesto, vamos a introducir un simple problema ficticio en donde las dependencias entre sus tareas sigan la secuencia de ejecución de la fabricación de una silla convencional. En su análisis, se detecta que el problema puede ser desglosado en cuatro tareas primitivas:

P = {respaldo, pata, asiento, componer}

Aunque el número total de tareas primitivas ha sido cuatro, en realidad, para el plan de trabajo (Ap) y para el componente de ejecutables (Ae) se contemplan siete: cuatro instancias de la tarea "pata" una para cada pata de la silla, una instancia de la tarea "respaldo", una de la tarea "asiento" y otra de la tarea "componer" (que se encarga de ensamblar todas las partes).

1	Componente1	Realiza la pata trasera 1
2	Componente1	Realiza la pata delantera 1
3	Componente1	Realiza la pata trasera 2
4	Componente1	Realiza la pata delantera 2
5	Componente2	Realiza el respaldo
6	Componente3	Realiza el asiento
7	Componente4	Compone todas las piezas

Tabla 1. Tareas para el plan "fabricar silla"

Como muestra la Tabla 1 (cuyo contenido es usado por el componente de ejecución Ae), cada tarea se ha codificado con un valor de tipo entero que será utilizado para describir el plan de ejecución y por los agentes para determinar el componente que debe ejecutar la resolución de cada tarea primitiva. Una tarea primitiva se corresponderá, por tanto, con la implementación de un componente software que realizará la tarea específica.

Incorporemos algunas restricciones al problema de fabricación para que el plan de trabajo resultante tenga algunas dependencias. Imaginemos que las cuatro instancias de la tarea "patas" de la silla son totalmente independientes, es decir, que se pueden realizar las cuatro al mismo tiempo. Sin embargo, vamos a suponer que para hacer el respaldo es necesario que estén hechas las dos patas traseras, y para hacer el asiento, las cuatro patas. Ante esta situación, y basándonos en la codificación de las tareas de la Tabla 1, en la Figura 2 se puede observar el grafo de dependencias entre las siete tareas descritas. Las puntas de flecha indican las dependencias entre tareas, de forma que, por ejemplo, la tarea 5 (fabricar respaldo) sólo se podrá ejecutar si y solo si han finalizado las tareas 1 y 3 (fabricar las patas traseras). Si generalizamos todo esto, podemos representarlo de la siguiente forma:

donde

T: Conjunto de tareas
D_i Conjunto de tareas dependientes de la tarea i
$D_i(j)$: Tarea que ocupa la posición j dentro del conjunto de tareas dependientes de la tarea i

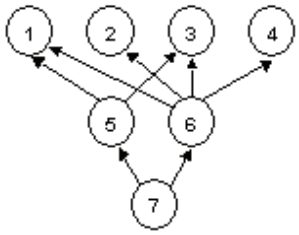


Figura 2. Grafo de dependencias para el ejemplo del problema de fabricación de sillas.

Una vez determinado el grafo de dependencias se genera el plan transformando esta representación gráfica a una colección de ternas de la forma:

$Ap = \{CT,TD,LT\}^N$

donde

CT: es el código de la tarea;
TD: es el número de tareas dependientes; y
LT: es la lista de tareas dependientes.

En el plan Ap resultante existirá una terna para cada tarea. Si una tarea i no tiene dependencias (tarea inicial), el campo TD se marca con el valor 0 y se deja vacía la lista LT. En el ejemplo anterior, obtenemos el siguiente conjunto de restricciones:

$$t1 \rightarrow 0$$
$$t2 \rightarrow 0$$
$$t3 \rightarrow 0$$
$$t4 \rightarrow 0$$
$$t5 \rightarrow (2, t1 \wedge t3)$$
$$t6 \rightarrow (4, t1 \wedge t2 \wedge t3 \wedge t4)$$
$$t7 \rightarrow (2, t5 \wedge t6)$$

Como se puede comprobar en el plan, las tareas t1, t2, t3 y t4 no tienen dependencias entre sí. Estas tareas se contemplan como tareas iniciales que se podrán ejecutar en paralelo usando el componente i establecido en la Tabla 1 para el componente de ejecución. Algo parecido sucede con las tareas t5 y t6. Aunque tienen tareas dependientes comunes (t1 y t3), esto no significa que t5 vaya a depender de t6, o viceversa. Solo lo sería si una estuviera en la lista de la otra, cosa que no ocurre en el ejemplo anterior. Por lo tanto, la ejecución de las tareas t5 y t6 también se puede paralelizar.

2.4 EL ESPACIO DE TRABAJO

El espacio de trabajo (ET) es un contenedor usado por los distintos ambientes de cómputo del sistema, donde se desarrolla la actividad de intercambio e interacción entre los componentes y los agentes del sistema. El espacio de trabajo se compone de siete elementos: (1) el componente de inicialización, AI; (2) el componente de ejecución, AE; (3) colección de componentes software que implementan las tareas finales, TE; (4) el plan de trabajo, AP; (5); la distribución de las tareas del plan a los agentes autónomos del sistema, AD; (6) el componente de sincronización, AS; y (7) una conjunto de testigos de habilitación de agentes, Ah. Así pues, el espacio de trabajo queda representado como un conjunto con siete clases de la forma:

$$Et = \{Ai, Ae, Te, Ap, Ad, As, Ah\}$$

El componente Ai realiza una configuración de los agentes del sistema. Todo agente necesita conocer otros cinco elementos: el plan, la distribución de tareas, detalles de sincronización, nombre de los componentes software de las tareas primitivas (véase Tabla 1), y el testigo que habilita/deshabilita un agente (Ah). De esta forma, el componente de inicialización Ai será una terna de entrada de cada agente de la forma:

$$Ai = \{Ap, Ad, As, Ae, Ah\}$$

Como se comentó en apartados anteriores, un agente permanece en estado de espera hasta que detecte la existencia de un testigo en el espacio de trabajo que active su secuencia normal de ejecución. Este testigo es el que se ha denominado como testigo de habilitación Ah. Todo agente tiene asignado un código distinto y estos testigos de habilitación son creados por el habilitador de agentes, Ha (véase Figura 1).

El componente de ejecución (Ae) usa la codificación de todas las tareas en las que se ha dividido el problema. El componente de ejecución es una terna de la siguiente forma:

$$Ae = \{Ct, Te, Ce, Co\}$$

donde

Ct: es un código de tarea;
Te: es el componente software de la tarea;
Ce: indica si la tarea necesita entradas o no; y
Co: es una breve definición de la tarea.

El plan de distribución Ad es una terna generada por el distribuidor del plan (DP) para asignar las tareas a cada uno de los agentes que están activos en el sistema. El proceso de distribución se describirá más adelante. Ad se forma como sigue:

$$Ad = \{CT, NA\}^N$$

donde

CT: es el código de la tarea, y
NA: es el número del agente.

En su instanciación, un agente recibe un código de identificación, código que debe aparecer en algunas ternas Ad, siempre y cuando se le haya asignado una parte del plan. Así pues, el agente i irá creando una lista interna de tareas cuyo código coincida con su código interno.

Considérese el plan de distribución que aparece en la Figura 3. Supóngase que, por cualquier motivo, el distribuidor de plan ha seleccionado sólo dos agentes para llevar a cabo la distribución, como se puede comprobar en el segundo campo de la tabla Ad de la figura. En este caso, el agente 1 detecta que se le han asignado las tareas {1, 2, 5, 7}y al agente 2 las tareas {3, 4, 6}. Seguidamente, y como veremos en el apartado dedicado al diseño de los agentes, cada uno de ellos utiliza esta lista interna de tareas, obtenida a partir de Ad, para "identificar" las dependencias del plan Ap y generar un subplan interno para cada agente (Figura 3).

Por otro lado, el componente de sincronización As coordina los

agentes para que resuelvan los planes parciales asignados a cada uno de ellos. Todos los agentes del sistema reconocen al componente de sincronización, que actúa de canal de comunicación controlando las tareas que han ido finalizado.

2.5 LOS AGENTES

Este apartado se complementa con el resto de las secciones descritas. Como se sabe, un agente permanece en espera hasta que detecte una señal (testigo Ah) que habilita su funcionamiento normal. Un agente del sistema está compuesto por varios módulos que en su conjunto resuelven una parte del plan global. Estos módulos se muestran en la arquitectura general de un agente de la Figura 4.

Figura 4. Arquitectura de un agente

El Detector del agente es el módulo que permanece a la espera de detectar la señal de habilitación. Esta señal (testigo Ah) procede del espacio de trabajo (ET): un contenedor comunicante de todos los componentes del sistema. Recibido el testigo, el Detector luego pasa el control al Actualizador y al Controlador. El primero obtiene el subplan y realiza una fase previa de configuración del agente para que éste pueda funcionar de forma correcta. El Controlador obtiene la información correspondiente del módulo anterior y procede a resolver el plan. Para comprobar el estado de las

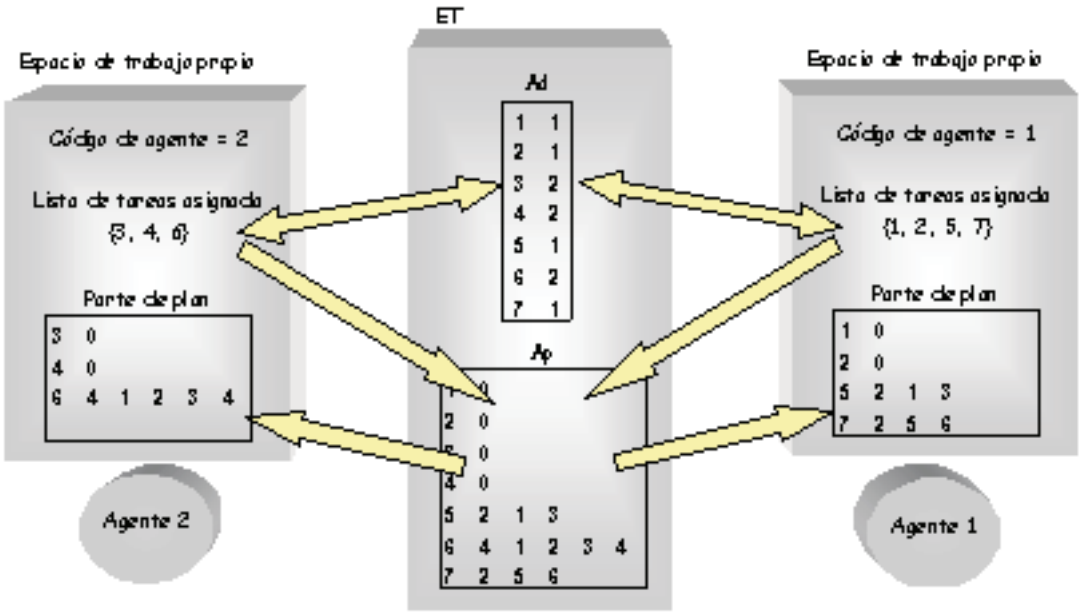


Figura 3. Influencia del plan de distribución Ad

dependencias de sus tareas con las del resto del sistema (reflejado en el plan) se utiliza el módulo Coordinador. Este coordinador establece un canal de comunicación a través del componente de sincronización As. Por último, una vez finalizada la ejecución del plan parcial (el asignado al agente), el Conmutador del agente se lo comunica al sistema por mediación de un testigo en el espacio de trabajo.

En la Figura 5 se muestra, a modo de ejemplo, uno de los algoritmos de comportamiento de un agente, concretamente se muestra el algoritmo del módulo Controlador. En la Tabla 2 se muestra un resumen de algunos de los tipos de datos utilizados por el Controlador. Este algoritmo, y el resto de ellos (no incluidos en este artículo) usan dos configuraciones básicas en su funcionamiento: los correspondientes a variables y tipos de datos locales al propio agente (al ambiente de cómputo donde éste se ubica) y los correspondientes a los remotos (ligados al espacio de trabajo).

Como se puede comprobar, el Controlador basa su comunicación en un mecanismo RPC: espera una llamada síncrona, obtiene el subplan, determina la colección de componentes software que ejecutarán las tareas, determina las dependencias existentes, resuelve el subplan y, en último lugar, inhabilita la sincronización RPC.

```
Algoritmo Controlador(cliente)
  REMOTA R; // configuración remota
  LOCAL L;    // configuración local
  PLAN LT;
  obtenerConfiguraciones(R,L);
  Mientras exista el agente
     esperar(R.habilitacion);   // wait
     inicializarPlan(LT);
     obtenerTareas(L.ambiente, R.distribucion, LT);
     obtenerComponentes(LT, R.componentes, L.destino);
     obtenerDependencias(R.plan, LT);
     resolver(LT, R.sincronizacion); // tareas iniciales
     finalizar(R.habilitacion);  // signal
  finMientras
finAlgoritmo
```

Figura 5. Algoritmo de control de un agente

TIPOS DE DATOS	
Tipo	**Comentario**
LOCAL:	Configuración local
ambiente	Código local del agente
inicio	Nombre del espacio de trabajo
destino	Nombre del recurso compartido
REMOTA:	Véase componente Ah
plan	
distribucion	
sincronizacion	
componentes	
habilitacion	

Tabla 2. Tipos de datos que usa un agente

3. EL PROCESO DE CONTROL

En este apartado se describen los componentes de control externos al espacio de trabajo (véase de nuevo la Figura 1). Una extensión de su aplicación se puede encontrar en [6]. En primer lugar, el componente GP acepta una representación del problema por parte del usuario y genera un plan de tareas a resolver. Para la representación del problema, el usuario puede utilizar alguna técnica de modelización del conocimiento. En nuestro caso, como se comentó en la sección donde se trató el plan, se ha utilizado una representación gráfica. Este componente acepta como entrada el plan (Ap) y genera un plan de distribución (Ad). Como se comentó, Ad especifica en qué ambientes (agentes) se tiene que ejecutar cada tarea. Este componente detecta las tareas iniciales (sin dependencias) y realiza una simulación de ejecución para determinar el grado de distribución del plan. Vamos a retomar el ejemplo seguido en apartados anteriores. Recordemos que el plan de ejecución a resolver es como el que se muestra en la Figura 6.

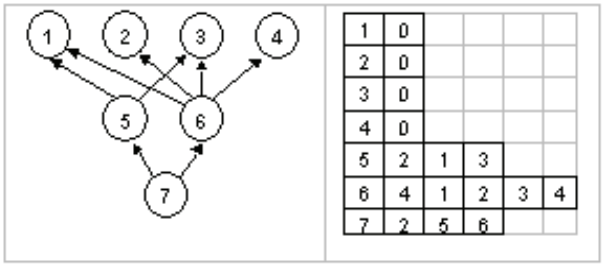


Figura 6. Plan de ejecución

Como se puede comprobar, en el plan existen cuatro tareas iniciales (1, 2, 3 y 4) que afectan directamente a las tareas 5 y 6 pero no a la 7. Igualmente, las tareas 5 y 6 afectan directamente a la tarea 7, finalizando, de esta forma, la ejecución del plan. Una correspondencia del grafo se ofrece, en forma de tabla, en la Figura 6. En las Figuras 7 y 8 se muestra un algoritmo de distribución de tareas (componente DP) y un plan de distribución de tareas, obtenido a partir de él.

El comportamiento del algoritmo de distribución es como sigue a continuación. En primer lugar, se detectan las tareas iniciales del plan, las cuales son fácilmente identificables en la tabla al no contener tareas dependientes, es decir, el segundo campo es un 0. Tras examinar dicha tabla, se comprueba que las cuatro tareas iniciales son 1, 2, 3 y 4. El algoritmo marca un primer nivel (N) para indicar que se pueden paralelizar hasta 4 tareas (nivel 1 = 4) anotando las tareas que son (véase Figura 8). Seguidamente, el algoritmo examina en la tabla las tareas que se ven afectadas por la ejecución de las del primer nivel. Se comprueba que esto sucede en las tareas 5 y 6, y procede a eliminarlas de la tabla, comprobando si estas (5 y 6) tienen dependencias. Como no las tienen, se marcan las tareas 5 y 6 dentro del nivel 2. A continuación, se marcan las líneas de la tabla donde aparezcan las tareas del nivel 2 (sólo la tarea 7). Se marcan el 5 y 6, y se eliminan. El nivel 3 solo contiene la tarea 7.

La dimensión de cada nivel determina el grado de paralelización. El grado de paralelización máximo viene impuesto por el nivel de mayor dimensión. Véase, a modo de ejemplo, la Tabla 3.

```
AlgoritmoDistribuidor(plan P)
  //nivel1 : tareas iniciales
  Para i de 1 a card(P)
    Si P(i,2) = 0 entonces N(1,j) ← P(i,1) y j++ finSi
  finPara
  k ← 1
  Mientras existan tareas ejecutar
    r ← 1 y i ← 1
    Si P(i,2) > 0 entonces
      Para j de 3 a card(P(i))+3
        Para n de 1 a card(N(k))
          Si P(i,j) = N(k,n) entonces
          eliminarElementoP(i,j)
          finSi
        finPara
      finPara
      Si card(P(i)) = 0 entonces
        N(k+1,r) ← P(i,1) yr++
      finSi
    finSi
    Si i = numeroTareas() entonces
      i ← 1 y k++ y r ← 1
      sino i++
    finSi
  finMientras
finAlgoritmo
```

Figura 7. Algoritmo de distribución

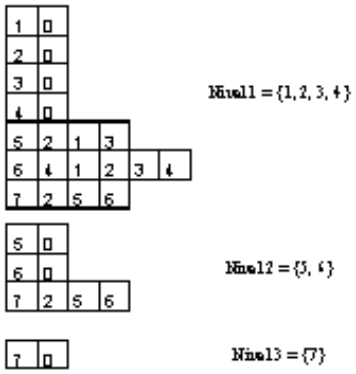


Figura 8. Niveles de ejecución

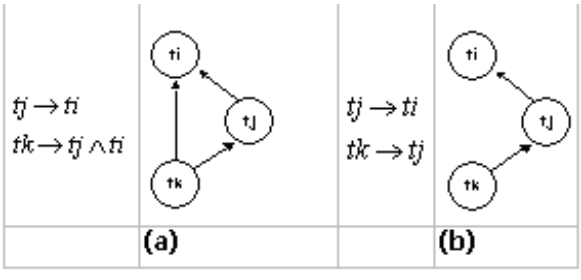


Figura 9. Dependencias entre tareas

Nivel	Dimensión	Tareas
1	4	1 2 3 4
2	2	5 6
3	1	7

Tabla 3. Niveles de distribución de tareas

A partir de la tabla se puede obtener el número (n) de agentes de la colección de k agentes disponibles en el sistema que pueden intervenir en el sistema (n<=k). En el ejemplo n=4 (dimensión máxima). El reparto de las tareas entre estos agentes se puede realizar de la siguiente forma: Agente1 = {1, 5, 7}, Agente2 = {2, 6}, Agente3 = {3}, Agente4 = {4}. Aun así, se pueden dar casos usuales en los que el grado de distribución sea superior al número total de ambiente de cómputo disponibles, es decir, que existan menos agentes disponibles que niveles de paralelización (n>=k). En este caso, una variante del algoritmo de distribución (no incluido en este artículo) consideraría la colección completa de los k agentes y, siguiendo un algoritmo de balanceado, realizaría un equilibrado de la carga de trabajo (las tareas) entre los agentes.

Por otro lado, el habilitador de agentes (HA) genera un testigo de habilitación para cada agente en espera que va a intervenir en la resolución del plan. El componente de optimización del plan (OP) se describe en el siguiente apartado.

4. ALGORITMO DE OPTIMIZACIÓN

Este apartado está estrechamente ligado con el apartado donde se trató el plan. El componente optimizador del plan OP es un módulo que acepta el plan de trabajo generado por el componente GP y elimina incongruencias de estado que pudieran existir entre tareas del plan. Estas incongruencias de estado podrían deberse a errores de análisis o diseño introducidos por el ingeniero a la hora de modelar el plan. Las incongruencias aparecen en partes del plan que cumplen las condiciones de la Figura 9 (a). Este fragmento de plan se interpreta de la siguiente forma: "la tarea tj se ejecutará si y sólo si se ha ejecutado ti, y la tarea tk se ejecutará si y solo si han finalizado las tj y ti".

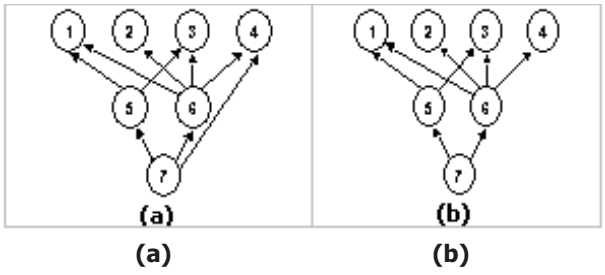


Figura 10. Dependencias entre tareas

Evidentemente, la dependencia directa que tiene la tarea tk respecto ti debería eliminarse del plan, existiendo tal dependencia entre tareas (tk, ti) de forma indirecta a través de la tarea tj. Por lo tanto, las tres tareas implicadas quedarían

de la forma que se muestra en la Figura 9 (b).

Estas dependencias innecesarias pueden no tener mucho sentido por separado, pero si se consideran planes de cierta envergadura donde el número de tareas es considerable, con dependencias del estilo que hemos visto, el funcionamiento del sistema podría ralentizarse debido a las comprobaciones de dependencias innecesarias; motivo de actuación del optimizador del plan.

El funcionamiento que tiene este componente se basa en el manejo interno de una matriz cuadrada que refleja el estado de las dependencias entre las tareas del plan. Además, el optimizador mantiene un conjunto de patrones de combinación (P) de dependencias correctas, que se obtienen al tomar muestras de tareas de tres en tres. Estos patrones se obtienen mediante matrices de 3x3 (Figura 13).

Para estudiar el método de optimización del plan nos centraremos en el ejemplo que hemos venido utilizando hasta el momento: el problema que sigue la secuencia de desarrollo de una silla. Para ello, incluiremos en el plan una dependencia no válida para que el optimizador la detecte: la dependencia que tiene la tarea 7 con la tarea 4 (Figura 10, a).

$$M = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Figura 10a. Optimización de dependencias

Para este caso, el optimizador OP aceptaría como válidas todas las dependencias del plan, salvo la nueva incluida, que la eliminaría quedando el plan como se muestra en la Figura 10 (b), como estaba. En primer lugar el Optimizador genera una matriz bidimensional de NxN, siendo N el numero de tareas del plan (para nuestro ejemplo N=7). Cada casilla de esta matriz almacena un valor igual a 1 o 0. Si la tarea i (fila) tiene una dependencia con la tarea j (columna), la casilla <i,j> contendrá un valor igual a 1, en caso contrario un 0. Aquellas filas de la matriz donde todas sus celdas sean un 0 serán tareas iniciales, es decir, tareas que no dependen de ninguna otra, y que se pueden, por tanto, ejecutar en paralelo. Para el ejemplo, la matriz resultante sería como la que sigue.

$$M = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix} \Rightarrow M(4,6,7) = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix}$$

Figura 11. Matriz resultante

En este caso, la mayor parte de las casillas de la matriz contienen el valor 0, ya que no existen muchas dependencias intermedias, la mayoría son tareas iniciales (1, 2, 3, 4). Por esta razón, las cuatro primeras filas de la matriz son 0. El optimizador seguidamente genera, a partir del plan, combinaciones de tareas de tres en tres, y elimina las dependencias incorrectas al hacer el producto matricial entre la matriz de la terna actual y los patrones preestablecidos P. Para el ejemplo, el optimizador comenzaría generando ternas de tareas a partir de la primera, es decir:

{(1,5,7), (1, 5, 3), (1, 6,2), (1, 6, 3), ...}

Todas las ternas de la lista se ordenan de menor a mayor, eliminando las repetidas. Así, por ejemplo, las ternas de la forma <1,6,3>, <3,6,1> y <6,1,3> serían consideradas iguales: <1,3,6>. Una vez generadas las ternas correctas, comienza el análisis de las independencias en cada una de ellas. Para una terna concreta, el proceso es como sigue:

1. Se obtiene la matriz de 3x3 asociada a la terna a partir de la matriz M.
2. Se calcula el producto matricial entre dicha matriz (3x3) y cada patrón válido.
3. Si la matriz resultante contiene un único valor 1, existe una independencia. Esta se elimina de la matriz general M.

El algoritmo general de optimización del plan se muestra en la Figura 12. Supongamos, por ejemplo, que el Optimizador ya ha obtenido la lista de todas las ternas posibles del plan, y que, actualmente, se está analizando la terna <4,6,7>, siendo su matriz asociada como la que sigue, M(4,6,7):

```
Algoritmo Optimizador(plan)
    P ← Preparar patrones validos
    M ← Obtener matriz plan
    T ← Obtener lista ternas
    Para i desde 1 a card(T)
        m_i ← obtenerMatriz(T_i)
        Para j desde 1 a card(P)
            r ← m_i × P_j
            Si totalUnos(r) = 1   //independ.
                (x, y) ← localizarPosicionUno(r)
                M(x, y) ← 0       //elimina dep.
            finSi
        finPara
    finPara
finAlgoritmo
```

Figura 12. Algoritmo de optimización del plan

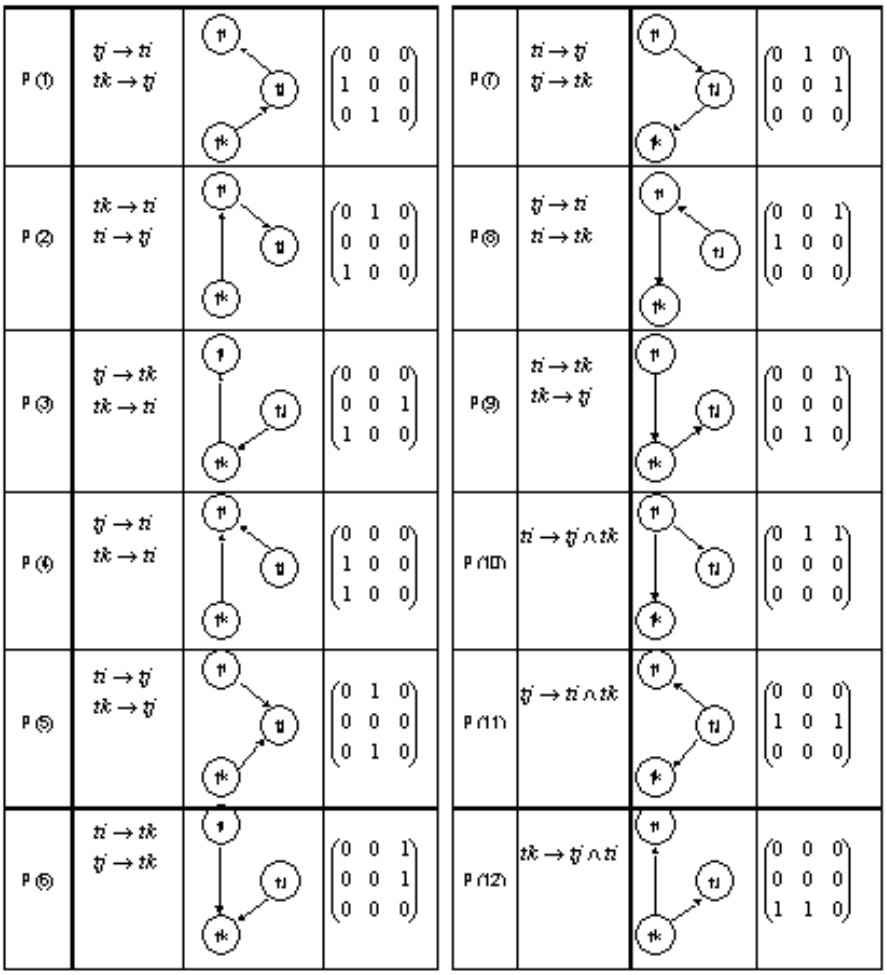


Figura 13. Patrones de dependencias válidas

El algoritmo a continuación efectúa el producto matricial entre M(4,6,7) y la colección de patrones de dependencias válidas P (Figura 13). En este caso se detecta una independencia al probar con P(1):

El único valor igual a 1 que se detecta dentro de la matriz R está en la posición (3,1), pero su posición relativa a la matriz M se corresponde con la (6,4), por lo que seguidamente se procede eliminar la dependencia en M cambiando el 1 por el 0.

5. CONCLUSIONES

La informática distribuida facilita la resolución de problemas complejos de ingeniería, donde la carga de cálculo es importante y las partes en las que se puede descomponer el problema están débilmente acopladas, i.e., son fácilmente desglosables (p.e., simulación por cálculo por elementos finitos). En este artículo se propone un marco de trabajo de un sistema basado en componentes y centrado en agentes que ofrece soporte para la resolución distribuida de problemas. El marco de trabajo ha sido implementado en C y CORBA. Las plataformas, en las que se establecen los ambientes de cálculo donde operan los agentes, han sido Linux (Fedora) y Windows 2000. Todas las pruebas realizadas han sido simbólicas, creando planes de trabajo ficticios (con/sin dependencias válidas). La implementación y las pruebas realizadas están disponibles en el sitio "http://www.ual.es/~liribarn/Investigacion/srdp". El marco de trabajo propuesto puede especialmente ser de utilidad en el ámbito de los sistemas de información geográficos (SIG) donde el volumen de información y la carga de cálculo es relativamente grande y variado y operan en tiempo real [5]. Un trabajo futuro pretende investigar la integración del marco propuesto con los SIG. Otro trabajo consiste en extender el algoritmo de distribución de agentes a uno descentralizado (en el artículo se propone uno centralizado); motivo por el cual no se han realizado pruebas de estimación de tiempos.

6. REFERENCIAS

[1] Bass. Component-Based Software Engineering. Putting the Pieces Together. (2001). Addison-Wesley.

[2] Crnkovic y Larsson. Building Reliable Component-Based Software Systems. (2002) Artech House Publishers.

[3] Durfee y Rosenshein. Distributed problem solving and multi-agent systems: Comparisons and examples. (1994), 94-104. In Proceedings of the Thirteenth International

Distributed Artificial Intelligence Workshop.
[4] Erman, Hayes-Roth, Lesser, y Reddy. The HearsayII Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty. (1980), 12 (2) /213-253 Computing Surveys.
[5] Gimblett. Integrating Geographic Information Systems and Agent-Based Modeling Techniques for Simulating Social and Ecological Processes. (2002). Oxford University Press.
[6] Iribarne, Bienvenido y Flores. Una metodología de distribución de procesos en problemas de simulación. (1999). 202-212, 2(4). Revista Computación y Sistemas.
[7] Labidi y Lejouad. De lìntelligence artificielle distribuée aux systèmes multi-agents. (1993). INRIA.
[8] OMG. The CORBA Component Model. Object Management Group (1999).
[9] Yang y Zhang. Application of MAS in Implementing Rational IP Routers on the Priced Internet. (Florida, 1997). 166-180. Springer-Verlag. Lecture Notes in Artificial Intelligence. LAIN, 1286.
[10] Zambonelli, Jennings y Wooldridge. Developing multiagent systems: The Gaia methodology. (2002). 317-370, 12(3). ACM Transactions on Software Engineering and Methodology (TOSEM).