ARTÍCULO

Autómatas Cooperativos Extendidos: sistemas multi-agente con dependencia geográfica

Carlos Herrero, Javier Oliver

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n VALENCIA 46022
{cherrero,fjoliver}@dsic.upv.es

Resumen

En este artículo se define una extensión del modelo de los Autómatas Cooperativos. Dicho modelo está destinado a la representación de sistemas distribuidos multi-agente. Pese a su gran capacidad expresiva, para determinados sistemas en apariencia simples, el modelo de los Autómatas Cooperativos resultante es muy complejo y hasta en ocasiones confuso. Para ilustrar esto se presenta un ejemplo que contiene un autómata cuyos estados dependen de su posición relativa en el sistema. Facilitar el diseño de este tipo de problemas es el objetivo de la extensión del modelo, basada en la idea de añadir un tipo especial de atributos. Asimismo se construye un algoritmo que muestra la equivalencia entre la extensión y el original conservando, por tanto, las características del modelo de los Autómatas Cooperativos y sus buenas propiedades.

Palabras clave: Concurrencia, Sistemas Multi-Agente, Autómatas Cooperativos, Redes de Petri.

1. Introducción

En [8] se presentó el modelo de los Autómatas Cooperativos [2], así como un ejemplo completo de su utilización. Los Autómatas Cooperativos (AC) permiten la representación de sistemas distribuidos multi-agente. Este modelo es especialmente adecuado para representar sistemas CSCW (Computer Suported Cooperative Work) [7, 4] y aplicaciones groupware [5]. Está basado en una doble representación mediante autómatas independientes que se coordinan a través de vectores de sincronización de acciones. Dependiendo del número de atributos de cada autómata, cada uno de los cuales describe la pertenencia a una tarea, podemos distinguir tres niveles de sincronización: el nivel básico o de *recursos*, sin atributos y en el que sólo está permitida la competencia entre consumidores, el nivel de *tareas*, con un único atributo que permite tareas y subtareas pero en el que no hay coordinación entre tareas distintas y, por último, el nivel de *relaciones*, en el que sí hay coordinación entre tareas y autómatas con dos o más atributos.

Un ejemplo de utilización de los AC para representar un sistema complejo puede verse en [8] y en [9] se muestra la simulación (de un modo básico) de otro modelo de representación de aplicaciones CSCW basado en autómatas: los autómatas Team [6, 11].

En este trabajo se presentan los Autómatas Cooperativos Extendidos (ACE) [9] que consisten en una versión de los Autómatas Cooperativos extendidos mediante la incorporación de un segundo tipo de atributos, llamados *atributos numéricos*, que demuestran ser especialmente útiles para la representación de sistemas en los que la ubicación

geográfica de un autómata cambia el diagrama de estados. Esta extensión y su evidente utilidad se muestra mediante un ejemplo sencillo. A continuación se introduce un algoritmo de conversión que permite traducir cualquier sistema modelado con Autómatas Cooperativos Extendidos en otro con Autómatas Cooperativos, de tal forma que ambos modelos resultan equivalentes y, por tanto, las buenas propiedades ya demostradas en el modelo original son aplicables al modelo extendido. Finalmente se presentan algunas conclusiones sobre el trabajo.

2. El modelo de los Autómatas Cooperativos

Vamos a presentar un formalismo cercano a las redes de Petri [10] para la modelización de la coordinación de agentes en un sistema distribuido denominado *Autómatas Cooperativos* [2]. Un sistema puede verse como una red de Petri de alto nivel en la que los tokens son elementos activos consistentes en un autómata junto con una porción de memoria privada. Las transiciones de la red son principalmente vectores de sincronización que dan todas las posibles interacciones entre objetos dinámicos. Se asume, básicamente, que los objetos pueden sincronizarse de acuerdo con su comportamiento (el servicio que prestan o, por el contrario, el servicio que solicitan) indiferentemente de su estado interno. Por ejemplo, si un proceso quiere imprimir un archivo, no tiene que preocuparse de qué impresora puede realizar el trabajo, cualquier objeto puede ser reemplazado por otro que ofrezca el mismo servicio de forma segura, incluso si ambos son muy diferentes respecto a otros servicios. Una sincronización puede requerir la creación de nuevos objetos (tokens) y puede también requerir la destrucción de otros.

El modelo básico de los *Autómatas Cooperativos* consiste en coordinación pura, es decir, los tokens pueden ser creados o destruidos a través de sincronizaciones de sus eventos, pero no pueden intercambiar información, puesto que no tienen memoria privada. Como instancias típicas de este modelo tenemos la *sincronización de autómatas* de Arnold y Nivat [1] y el formalismo *Gamma* [3].

Este modelo básico puede ser extendido, sin embargo, para tratar *Workflow Management Systems*, que describen el problema de instancias en un sistema distribuido. Por ejemplo, una instancia puede estar involucrada con la participación de un investigador en una conferencia; entonces, se crean varias tareas, por ejemplo: la inscripción, la reserva del vuelo, del hotel, etc. Estas tareas requieren la coordinación de varios sistemas de computadores que pertenecen respectivamente a la institución del investigador, a la agencia de viajes, a la organizacion de la conferencia, etc. A cada instancia se le debe asignar un único identificador que se transmite a cada una de las tareas que se generan para que el conjunto del sistema distribuido pueda reconocer las tareas involucradas con el tratamiento de una instancia dada. Se modelan las tareas con tokens que, por lo tanto, contienen un identificador en su memoria privada. En semejante sistema, sin embargo, todavía no hay flujo de información entre tokens diferentes, debido al hecho de especificar un vector de sincronización que permite afirmar tan solo que ciertos componentes pueden tener el mismo identificador.

Avanzando un paso más, podemos considerar aplicaciones *groupware*, que involucran sistemas distribuidos de agentes que cooperan para resolver una tarea global y cuya estructura puede cambiar dinámicamente. En este caso, un token puede contener también referencias a otros tokens en su memoria privada. Dado que en estas aplicaciones los tokens incorporan tanto estructuras de datos como especificaciones de comportamiento, pueden ser considerados como objetos en el sentido de la terminología de los lenguajes orientados al objeto.

Si la memoria local se presenta como una lista de atributos-valores parece haber una delimitación clara de la expresividad en términos del flujo de información que puede ser medida por el número de atributos. En [2] se muestra que la expresividad completa se alcanza tan pronto se admitan dos atributos. La jerarquía de los Autómatas Cooperativos consta entonces de tres clases: la clase completa (con dos o más atributos) que se adapta adecuadamente al diseño de aplicaciones *groupware*, la clase intermedia (con un atributo) que se adapta a las necesidades de las aplicaciones *workflow* y el modelo básico (sin atributos) que permite la coordinación pura de eventos sin flujo de información.

La clase completa de autómatas cooperativos es suficientemente potente como para permitir la simulación de máquinas de Turing. No hay, por lo tanto, esperanza alguna de obtener herramientas generales automáticas para verificar propiedades

de estos sistemas; sin embargo, este modelo general es útil como herramienta de modelado.

Por el contrario, se puede decidir si un sistema tiene un número finito de estados si lo restringimos a la así llamada subclase de autómatas cooperativos *workflow* para la que tan solo se permite un atributo. Esta propiedad, también llamada *boundedness (acotamiento)*, está demostrada fundamental para la verificación de sistemas *workflow*.

2.1. Una aproximación al modelado

El modelado de sistemas utilizando los *Autómatas Cooperativos* generalmente requiere los siguientes pasos:

1. Identificar al autómata en el sistema. Deben ser agentes (como procesos u objetos) o artefactos que sirvan para mantener las relaciones entre autómatas.
2. Identificar los estados de cada autómata y las acciones que conllevan. No es necesario identificar todas las acciones de los autómatas en este punto.
3. Identificar las interacciones entre autómatas. En el modelo, dichas interacciones están representadas como vectores de sincronización. Cuando en una interacción todos los estados asociados cambian, en sus respectivos autómatas, lo hacen de manera síncrona. Las interacciones pueden provocar la creación o destrucción de autómatas.
4. Iterar estos pasos si es necesario.
5. Finalmente, es importante validar el modelo cuando está completo. ¿Refleja adecuadamente el modelo al sistema bajo desarrollo? Si es así, entonces, ¿es el sistema lo que se deseaba representar? El modelo puede servir para garantizar que las descripciones son correctas.

A continuación se perfila la aproximación al diseño e implementación de sistemas cooperativos, que está modelado con *Autómatas Cooperativos*.

Cuando se emprende el diseño de un sistema cooperativo, la primera etapa consiste en identificar los recursos a utilizar, las tareas a realizar usando dichos recursos y los agentes del mundo exterior responsables de las tareas. Estos últimos, en contraste con los recursos y tareas que habitualmente existen en el sistema, deben permitir actuar a través de artefactos que proporcionen conexiones para establecer relaciones entre ellos y para controlar las tareas que inician. Si tenemos en mente el diseño y la evaluación temprana de un modelo de un sistema cooperativo, y no la realización completa de los agentes de control del mundo exterior, esta diferencia desaparece dado que tanto los recursos como las tareas se representan como artefactos en el modelo. Sin embargo, quedan unas leves diferencias entre recursos, tareas y agentes. Los recursos son estáticos (no tienen creación dinámica ni se dividen) y son intercambiables (todos los elementos de una fuente de recursos capaces de realizar el mismo servicio en un momento dado son equivalentes). En contraste con los recursos, las tareas son objeto de creación dinámica y están habitualmente divididas en subtareas que se reagrupan más tarde si es necesario (esto es un aspecto típico de los sistemas de manufactura y de flujos de trabajo). Además, una tarea tiene un tiempo de vida y aparece como una colección dinámica de subtareas que pueden involucrar diferentes recursos. Esto es de vital importancia en un sistema cooperativo para añadir información permanente a subtareas, permitiéndoles reconocer su tarea principal. El caso de agentes externos, que pueden también participar y desaparecer del sistema en ciertos momentos, es diferente. El problema no es tanto mantener la traza de grupos de agentes, como mantener datos operativos que representen sus relaciones conocidas en el sistema, basadas en que el sistema puede establecer reglas de interacción. Estos datos deben ser proporcionados mediante grafos o hipergrafos (con agentes como nodos), modificados de acuerdo a las reglas específicas de transformación cuando el sistema cambia las relaciones entre agentes (como resultado de alguna interacción).

2.2. Autómatas Cooperativos

A continuación presentamos la definición formal de los Autómatas Cooperativos.

Hipótesis 1: El comportamiento autónomo de un agente puede ser descrito por un autómata finito.

Hipótesis 2: Hay un número finito de tipos de agentes y, por lo tanto, de autómatas.

Si se añade a todos los autómatas un estado ficticio $\perp$ con transiciones entrantes y salientes modelizando la creación y destrucción de agentes, se obtiene que el comportamiento de los agentes puede ser descrito por un autómata agente.

Hipótesis 3: El estado de un sistema es un multiconjunto de tokens, que representan a los agentes "vivos" con sus respectivos estados, más un token en el estado ficticio $\perp$, como una fuente de agentes.

Hipótesis 4: Las relaciones entre agentes están codificadas mediante registros anexados a los tokens agente. Todos los registros especifican los valores de una lista fija de atributos. El valor asignado a los atributos es arbitrario, pero la comparación de valores de atributos es significativa.

Hipótesis 5: Los cambios de estado resultan de las transacciones síncronas. Parte del control de transacciones consiste en comprobar y modificar las relaciones entre agentes.

Hipótesis 6: Hay un número finito de tipos de transacciones.

Las hipótesis previas permiten la siguiente definición:

Definición 1 ***(Autómatas Cooperativos)*** *Un sistema de autómatas cooperativos (CA) es una quintupla* (Σ, S, T, Att, R) *donde* (S, Σ, T) *es un autómata agente(AA), con un conjunto finito de estados* S*, un conjunto finito de acciones* Σ *y un sistema de transición* $T \subseteq S_\perp \times \Sigma \times S_\perp$ *donde* $S_\perp = S \cup \{\perp\}$ *y* $\perp$ *es el estado inicial, Att es un conjunto finito de atributos y* R *es un conjunto finito de reglas de transacción. Cada regla* $r = (\mathbf{synch}, \mathbf{guard}, \mathbf{update})$ *consiste en un vector* **synch** *de acciones de sincronización, requerido por los agentes respectivos que se involucran en la transacción, una guarda* **guard** *especificando las relaciones que deben mantenerse entre ellos para permitir la transacción y una actualización* **update** *de las relaciones en la compleción de la transacción.*

- *Un* ***vector de sincronización*** *es un vector* $\mathbf{synch} = A_1 \cdot a_1 + ... + A_n \cdot a_n$ *donde* a_i *son nombres de acción* ($a_i \in \Sigma$) *y* A_i *son nombres de agentes con el rango limitado a la regla (limitado en cada instancia a los tokens agente con estados locales* s_i *que producen acciones* a_i*).*
- *Una guarda* **guard** *es una conjunción de restricciones de igualdad (respectivamente, desigualdad)* $A_i \cdot u = A_j \cdot v$ *o* $A_i \cdot u = 0$ *(respectivamente,* $A_i \cdot u \neq A_j \cdot v$ *o* $A_i \cdot u \neq 0$*) donde* a_i *y* a_j *son acciones regulares o de destrucción en* ${}^\bullet r = \{A_i | a_i \in \Sigma_2 \cup \Sigma_3\}$ *y* u *y* v *son atributos.*
- *Una actualización* **update** *es una secuencia finita de asignaciones* $A_i \cdot u := A_j \cdot v$ *o* $A_i \cdot u :=$ **new** *donde* a_i *es una acción de creación o regular en* $r^\bullet = \{A_i | a_i \in \Sigma_2 \cup \Sigma_3\}$*,* u *y* v *son atributos tales que no ocurren asignaciones múltiples.*

Un autómata token es un par $x = (s, \rho)$ donde $s \in S$ es el estado del token x (escrito como $x \cdot estado = s$) y $\rho : Att \to \mathbb{N}$ guarda los valores $x \cdot u = \rho(u)$ de los atributos u del token x. Se han escogido valores enteros para los atributos, con 0 como valor por defecto, pero cualquier otro conjunto enumerable habría servido también. Un estado del sistema o *marcado* es un multiconjunto finito de tokens. El disparo en un marcado M de una regla $r = (\mathbf{synch}, \mathbf{guard}, \mathbf{update})$ con un vector de sincronización $\mathbf{synch} = A_1 \cdot a_1 + ... + A_n \cdot a_n$ lleva a un marcado $M' = M - L + R$, donde $L = \{x_i | A_i \in {}^\bullet r\}$ y $R = \{x_i' | A_i \in r^\bullet\}$ son los multiconjuntos de tokens, respectivamente, consumidos o generados por el disparo.

Activación de una regla

Un vínculo $\{A_i = x_i | A_i \in {}^\bullet r \wedge x_i \in M\}$ activa r si y sólo si:

1. $L = \{x_i | A_i \in {}^\bullet r\}$ es más pequeño que M;
2. $x_i \cdot estado \xrightarrow{a_i} s_i$ en AA cuando $a_i \in \Sigma_2$;
3. $x_i \cdot estado \xrightarrow{a_i} \perp$ en AA cuando $a_i \in \Sigma_3$;
4. la guarda es válida bajo esa ligadura.

Evolución sincronizada de un agente

El disparo de r bajo una ligadura activada supone:

1. La producción de tokens agente x_i' tales que $\perp \xrightarrow{a_i} x_i' \cdot estado$ en AA para todas las acciones de creación $a_i \in \Sigma_1$;

2. La transformación de los tokens x_i en tokens x'_i tales que $x_i \cdot estado = s_i$ para todas las acciones regulares $a_i \in \Sigma_2$;

3. El borrado de los tokens agente x_i para todas las acciones de destrucción $a_i \in \Sigma_3$.

Actualización de relaciones

Los atributos $u, v, ...$ de los tokens producidos en $R = \{x'_i \mid A_i \in r^\bullet\}$ junto con los valores respectivos $x'_i \cdot u, x'_i \cdot v, ...$ son tales que, cuando $A_i = x_i$ para $A_i \in {}^\bullet r$ y $A'_i = x'_i$ para $A_i \in r^\bullet$, debe mantenerse lo siguiente:

1. $A'_i \cdot u = A_j \cdot v$ cuando $A_i \cdot u = A_j \cdot v$ está en **update** y $A_j \in {}^\bullet r$;

2. $A'_i \cdot u = A_i \cdot u$ cuando $A_i \in {}^\bullet r \cap r^\bullet$ y A_i no está asignado en **update**;

3. $A'_i \cdot u = 0$ cuando $A_i \in r^\bullet \backslash {}^\bullet r$ y A_i no está asignado en **update**;

4. Todos los valores $A'_i \cdot u$ tales que $A_i \cdot u := new$ está en **update** son diferentes (para cualquier i y u) y difieren de los valores de los atributos de todos los tokens de M;

5. $A'_i \cdot u = A'_j \cdot v$ cuando $A_i \cdot u = A_j \cdot v$ está en **update** y $A_j \in {}^\bullet r \cap r^\bullet$.

No se ha definido explícitamente la ejecución concurrente de múltiples instancias de reglas de transacción múltiples con vínculos independientes; sin embargo, esta extensión está en proyecto.

De la hipótesis 1, el modelo abstrae el tratamiento funcional de datos contenidos por los agentes y centrados en el control, usando los nombres de acción como abreviaturas para los métodos que operan sobre los propios datos. De las hipótesis 4 y 5, el modelo abstrae los datos distribuidos que representan las relaciones inter-agentes y que no pueden ser operados por los agentes. Estos datos protegidos (atributos con su valor) son las propiedades del controlador abstracto descrito por las reglas de transacción. De la hipótesis 5, el modelo abstrae las transacciones síncronas, aunque éste no sea el caso en muchos sistemas reales.

Estas hipótesis facilitan el uso de un simulador del modelo donde los estados puedan visualizarse como una tabla de tokens o como un grafo de relaciones. Esto permite también la traducción de los modelos a un código ejecutable (prototipo del sistema) donde los autómatas agente se expanden a objetos distribuidos que pueden interactuar con las reglas de transacción.

Un sistema marcado de autómatas cooperativos es un sistema de autómatas cooperativos con un marcado inicial. Usualmente se especifica el marcado inicial a través de una regla específica llamada *regla de inicialización*, con una parte condicional cuyo disparo produce el marcado inicial. Esta regla está siempre activa, pero está dispuesta para ser disparada sólo inicialmente.

3. Autómatas Cooperativos Extendidos (ACE)

Tal y como ya aparece en [8], con los Autómatas Cooperativos se pueden modelizar sistemas distribuidos, CSCW y aplicaciones groupware; sin embargo, en ocasiones es necesario el uso de modelos muy complejos para mostrar situaciones en apariencia simples. Este es el caso de aquellos sistemas en los que el conjunto de estados depende de algún valor externo como, por ejemplo, la posición geográfica. Para ilustrar esto, se presenta un ejemplo sencillo. Supongamos que se desea representar un sistema en el que el conjunto de estados posibles (y, por tanto, de acciones posibles) de uno de los autómatas del sistema cambia según la posición. La solución en el modelo de los Autómatas Cooperativos precisa distinguir como estados diferentes aquéllos que tienen lugar en posiciones distintas. Veamos un ejemplo de un sistema de estas características.

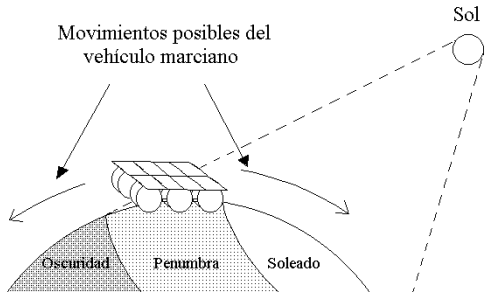

Figura 1. Posiciones posibles y Movimientos del MSL

El prototipo de la "sonda solar marciana" (Figura 1) Mars Solar Lander (MSL) realiza tres funciones

principales: cargar las baterías con energía solar, tomar fotografías infrarrojas y recoger muestras de roca. Sin embargo, estas tres funciones no pueden realizarse en cualquier lugar. Si trazamos una linea imaginaria que traspase las zonas de sol, penumbra y sombra, la tarea de recoger muestras sí puede realizarse siempre, pero la de tomar fotografías infrarrojas sólo puede hacerse en las zonas de penumbra y de sombra, pues en la de sol las fotografías saldrían veladas; asímismo, la tarea de recarga de baterías sólo puede realizarse en las zonas de sol y de penumbra. Por tanto, el autómata puede realizar las siguientes acciones partiendo de la posición *light* en donde puede resolver las tareas de recogida de rocas y de recarga, o moverse a la posición *half-light* donde puede realizar las tres tareas, o moverse finalmente a la posición *shadow* donde sólo puede recoger rocas o tomar fotografías pero no cargar baterías. Las acciones que representan el cambio de estado son del tipo "go_X_Y_Z", es decir, *ir del lugar X al lugar Y a realizar la tarea Z* o bien acciones "Z", esto es, *realizar la tarea Z sin cambiar de lugar* o, finalmente, acciones "return_Y_X_Z" que indican *volver del lugar Y al lugar X a realizar la tarea Z*. Está claro que por cada cambio posible tenemos una acción asociada. En la Figura 2 se ha representado el diagrama de estados así como las transiciones posibles entre ellos. No se muestra cada acción puesto que el elevado número de posibilidades haría ilegible el diagrama. Para clarificar el diagrama, se han representado los distintos estados como parejas (tarea, lugar) siendo las tareas *Collect, Charge y TakeP* y etiquetando los lugares como p1, p2 y p3 (zona de sol, de penumbra y de sombra, respectivamente).

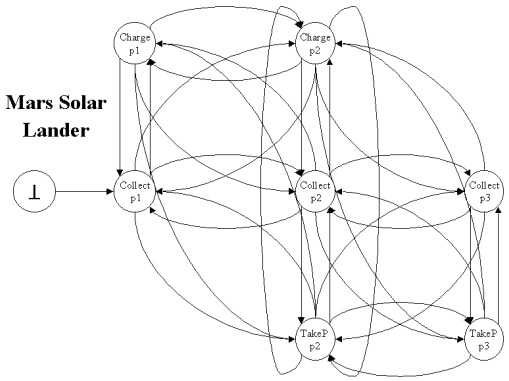

Figura 2. Ejemplo de MSL con un Autómata Cooperativo

Es evidente la complejidad de este autómata. Además, para individualizar las posibles acciones es preciso definir una regla para cada una de ellas y ello sin contar con el resto de agentes que pudiera haber en el sistema y con quienes también haría falta sincronización. Sólo contando con las acciones simples que pueden verse en este autómata se tiene un sistema de 34 reglas.

¿Cómo podría simplificarse este modelo? Una respuesta a esta pregunta consiste en considerar la posibilidad de utilizar un tipo especial de atributos para representar la posición del MSL: atributos numéricos. No obstante, estos atributos plantean a su vez nuevos interrogantes. ¿Estos atributos numéricos tendrán las mismas características que los atributos clásicos de los Autómatas Cooperativos? ¿Indican la pertenencia a una tarea? ¿Siguen las mismas reglas?

Estos nuevos atributos deben tener un papel completamente distinto al de los anteriores. Para las cuestiones relacionadas con la sincronización entre tareas sólo se utilizarán los atributos originales. Si se quiere mantener las características deseables de los autómatas cooperativos en los autómatas cooperativos extendidos, hay que limitar de forma drástica el papel de los atributos numéricos, considerándolos como información adicional del estado del autómata y manteniendo, por tanto, los tres niveles de autómatas (sin atributos, con un atributo y con dos o más atributos) referidos únicamente a los atributos identificadores sin tener en cuenta la existencia o no de los atributos numéricos. Sin embargo, dado que estos atributos numéricos son modificables en tiempo de ejecución, la complejidad del árbol de derivación de estados se incrementará considerablemente. Si se quiere, por tanto, mantener la decibilidad de propiedades del modelo original es necesario limitar el conjunto de valores posibles de los atributos numéricos y exigir que sean de tipo entero y dentro de un rango finito de valores.

Así pues, se considera la existencia de guardas implícitas que limitan el rango de valores de estos nuevos atributos. Para simplificar el modelo no se incluirán estas guardas implícitas en las reglas, pero sí se considerarán como necesarias para permitir el disparo de una regla.

La definición del Sistema de Autómatas Cooperativos Extendido, consiste básicamente en la inclusión del segundo tipo de atributos y en la redefinición de guardas y actualizaciones en cada regla incluyendo las restricciones y acciones posibles para estos nuevos atributos.

Definición 2 *Un* Sistema de Autómatas Cooperativos Extendido (ACE) *es una quintupla* $(\Sigma, S, T, (Att_1, Att_2), R)$ *donde* (Σ, S, T) *es un autómata agente (AA),* Att_1 *es un conjunto finito de atributos identificadores (enteros) y* Att_2 *es un conjunto finito de atributos numéricos en el cual* $\forall u \in Att_2, -\varsigma < u < \tau$ *con* $\varsigma, \tau \in \mathbb{N}$ *y tal que* $Att_1 \cap Att_2 = \emptyset$ *y* R *es un conjunto finito de reglas de transición. Cada regla* $r =$ (synch, guard, update) *consta de un vector* synch *de acciones sincronizadas requeridas por los agentes respectivos involucrados en la transacción, una guarda* guard *que muestra las relaciones que deben mantener los atributos para poder realizar la transacción y una actualización* update *de las relaciones al completarse la transacción.*

- *Un vector de sincronización es un vector* $synch = A_1 \cdot a_1 + ... + A_n \cdot a_n$ *donde* a_i *son nombres de acción* $(a_i \in \Sigma)$ *y* A_i *son nombres de agentes, con ámbito limitado a la regla (es decir, limitado en cada instanciación a tokens agente con estados locales* s_i *y acciones* a_i*).*
- *Una guarda es una restricción (o restricciones) que depende del tipo de atributo:*

  si $u, v \in Att_1$ *entonces* $A_i.u = A_j.v$ *(o* $\neq$*),* $A_i.u = default$ *(o* $\neq$*)*

  si $u, v \in Att_2$ *entonces* $A_i.u = A_j.v$ *(o* $\neq$*),* $A_i.u = const$ *(o* $\neq$*),* $A_i.u \geqslant A_j.v$ *(o* $<$*),* $A_i.u > A_j.v$ *(o* $\leqslant$*)*

  donde a_i *y* a_j *son acciones regulares o de destrucción en* $^\bullet r = \{A_i | a_i \in \Sigma_2 \cup \Sigma_3\}$ *y* u*,* v *son atributos.*
- *Una actualización es un conjunto finito de asignaciones (en el caso de atributos identificadores) u operaciones simples y asignaciones (en el caso de atributos numéricos):*

  si $u, v \in Att_1$ *entonces* $A_i.u := A_j.v$*,* $A_i.u := new$

  si $u, v \in Att_2$ *entonces* $A_i.u := A_j.v$*,* $A_i.u := const$*,* $A_i.u += const$*,* $A_i.u -= const$*,* $A_i.u := new$

  donde a_i *y* a_j *son acciones regulares o de creación en* $r^\bullet = \{A_i | a_i \in \Sigma_1 \cup \Sigma_2\}$ *y* u*,*v *son atributos tales que no hay asignaciones múltiples. Se asume que en las actualizaciones de los atributos numéricos se mantienen las restricciones que limitan los valores posibles.*

Es decir, este nuevo tipo de atributos puede intervenir en las guardas (limitando el disparo de las reglas como formando parte de los estados) y ser actualizado mediante asignaciones o incrementos con constantes.

Según este nuevo modelo, la Figura 3 muestra el diagrama de estados resultante del ejemplo anterior.

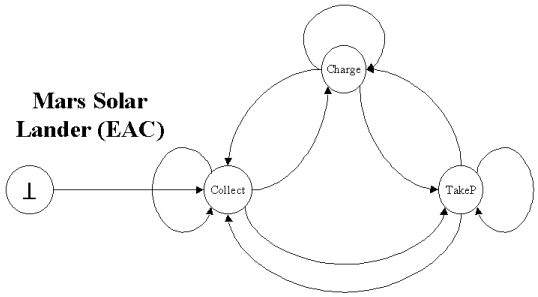

Figura 3. El autómata MSL versión ACE

La Figura 3 sólo muestra acciones que relacionan el paso de una actividad a otra independientemente de la situación geográfica del autómata. La situación geográfica aparece en las reglas de transición que se definen a continuación. Internamente, el autómata MSL tiene un atributo numérico denominado "place" para identificar su posición. El rango de valores posibles es {0,1,2} y se usa para limitar las acciones posibles en las guardas.

name: Collect_to_Charge_stopped
  synch: MSLander.co_ch()
  guard: MSLander.place $\leqslant$ 2
  update: null

name: Collect_to_TakeP_stopped
  synch: MSLander.co_ta()
  guard: MSLander.place $\geqslant$ 2
  update: null

name: Collect_to_Charge_going
  synch: MSLander.co_ch()
  guard: MSLander.place = 1
  update: MSLander.place += 1

name: Collect_to_Charge_returning
  synch: MSLander.co_ch()
  guard: MSLander.place $\geqslant$ 2
  update: MSLander.place -= 1

name: Collect_to_TakeP_going
  synch: MSLander.co_ta()
  guard: MSLander.place $\leqslant$ 2
  update: MSLander.place += 1

name: Collect_to_TakeP_returning
synch: MSLander.co_ta()
guard: MSLander.place = 3
update: MSLander.place -= 1

name: Collect_to_Collect_going
synch: MSLander.co_co()
guard: MSLander.place $\leqslant$ 2
update: MSLander.place += 1

name: Charge_to_Collect_returning
synch: MSLander.ch_co()
guard: MSLander.place = 2
update: MSLander.place -= 1

name: Charge_to_Charge_going
synch: MSLander.ch_ch()
guard: MSLander.place = 1
update: MSLander.place += 1

name: Charge_to_Charge_returning
synch: MSLander.ch_ch()
guard: MSLander.place = 2
update: MSLander.place -= 1

name: TakeP_to_Collect_stopped
synch: MSLander.ta_co()
guard: null
update: null

name: TakeP_to_Charge_stopped
synch: MSLander.ta_ch()
guard: MSLander.place = 2
update: null

name: TakeP_to_Collect_going
synch: MSLander.ta_co()
guard: MSLander.place = 2
update: MSLander.place += 1

name: TakeP_to_Collect_returning
synch: MSLander.ta_co()
guard: MSLander.ubicacion $\geqslant$ 2
update: MSLander.place -= 1

name: TakeP_to_TakeP_going
synch: MSLander.ta_ta()
guard: MSLander.place = 2
update: MSLander.place += 1

name: TakeP_to_TakeP_returning
synch: MSLander.ta_ta()
guard: MSLander.place = 3
update: MSLander.place -= 1

Como puede observarse, con este sistema tenemos que las reglas son 22 frente a las 34 del modelo original y el diagrama de transición de estados es, evidentemente, más sencillo y clarificador.

Si podemos demostrar que el modelo extendido es equivalente al original, tendremos las características deseables de aquél, más la facilidad de representación de éste. Este es el objetivo de la siguiente sección.

4. Transformación de un sistema ACE en uno AC

La propiedad de acotamiento en el modelo de los Autómatas Cooperativos es decidible en el nivel de tareas, que contiene únicamente autómatas con, como máximo, un atributo. La demostración de este hecho puede verse en [2]. No obstante, aparecen problemas adicionales si se habla del modelo extendido, puesto que el árbol de derivación ya no depende sólo de los atributos identificadores sino también de los atributos numéricos. Demostrando que todo sistema extendido es a su vez una versión comprimida (más legible) de un sistema de Autómatas Cooperativos y, sabiendo que un sistema original es por definición también uno extendido, se pueden trasladar las características del segundo sobre el primero. Se muestra la equivalencia de ambos sistemas considerando cada posible valor de los atributos numéricos como parte del grafo de estados y aplicando el siguiente algoritmo.

Algoritmo:

1. Repetir:
   - Para toda regla R en cuya guarda aparezca una expresión sobre un atributo numérico del tipo $A.u == B.v$, $A.u\ != B.v$, $A.u >= B.v$, $A.u > B.v$, $A.u < B.v$ o $A.u <= B.v$ y/o una actualización del tipo $A.u = B.v$, $A.u += B.v$, $A.u -= B.v$, (donde A, B son nombres de autómatas y u, v son atributos numéricos de dichos autómatas respectivos) es decir, aquellas expresiones que en la parte izquierda no tengan una constante sino un valor de atributo numérico,
     - generar tantas reglas como valores posibles tenga el rango del atributo v en el autómata B, de manera

que en la parte izquierda de las reglas que contenían el atributo $B.y$ sólo aparezcan constantes,

- eliminar la regla que hemos expandido en las anteriores,
- eliminar las reglas que aparezcan con guardas imposibles (por ejemplo, $A.x == 5 + A.x == 4$)

hasta que no queden atributos numéricos en la parte izquierda de una guarda o actualización.

2. Para cada autómata con atributos numéricos hacer:

   Sea $\Sigma = \{s_1, ..., s_m\}$ el conjunto de acciones del autómata y $R = \{r_1, ..., r_p\}$ el conjunto de reglas en el que se ve involucrado.

3. Para cada atributo numérico u, suponiendo $min \leq u \leq max$ con min, $max \in Integer$:

4. Construir, con el conjunto de estados $Q = \{\perp\} \cup \{q_1, ..., q_n\}$, un nuevo conjunto de estados Q' de un nuevo autómata AC sin el atributo u como sigue: $Q' = \{\perp\} \cup \{q_{1_{\text{mín}}}, ..., q_{n_{maz}}, ..., q_{1_{\text{mín}}}, ..., q_{n_{\text{máx}}}\}$.

5. Sea $\Sigma' = \Sigma'_1 \cup \Sigma'_2 \cup \Sigma'_3$ como sigue, $\Sigma'_1 = \{s'_\perp\}$ donde $\perp \xrightarrow{s_\perp} q_{ini_k}$ es el estado inicial correspondiente a la inicialización del atributo, $\Sigma'_2 = \emptyset$ y $\Sigma'_3 = \emptyset$.

6. Sea $R' = \{r'_{ini}\}$ donde r'_{ini} se construye cambiando en r_{ini} la acción $s_\perp$ con la acción $s'_\perp$ en la regla inicial.

7. Para cada acción s_i, que se ejecute en una regla r_j donde el atributo no aparezca en la guarda ni en la actualización:

   - añadir al conjunto Σ'_1 la acción s'_i tal que si teníamos la transición $q_a \xrightarrow{s_i} q_b$ ahora tenemos $q_{a_k} \xrightarrow{s'_i} q_{b_k} : \forall k : min \leq k \leq max$,

   - añadir a R' la regla r'_j obtenida tras cambiar en r_j la acción s_i con s'_i.

8. Para cada acción s_i, que se ejecute en una regla r_j donde el atributo no aparezca en la guarda pero sí en la parte izquierda de la actualización:

   - añadir al conjunto Σ'_2 la acción s_i^α tal que si teníamos la transición $q_a \xrightarrow{s_i} q_b$ ahora tenemos $q_{a_k} \xrightarrow{s_i^\alpha} q_{b_{k+\alpha}} : \forall k : min \leq k \leq max - \alpha$ (o respectivamente, $\forall k : min + \alpha \leq k \leq max$),

   - añadir a R' la regla r'_j obtenida tras cambiar en r_j la acción s_i con s_i^α.

9. Para cada acción s_i, que se ejecute en una regla r_j donde el atributo aparezca en la guarda como $== x$ o $! = x$:

   - añadir al conjunto Σ'_3 la acción $s_{i_{==x}}$ (o, respectivamente, !=x) tal que si teníamos la transición $q_a \xrightarrow{s_i} q_b$ ahora tenemos $q_{a_x} \xrightarrow{s_{i==x}} q_{b_{x+\alpha}}$ (o, respectivamente, !=x) donde α es el valor de la constante añadida en la actualización (si es que existe),

   - añadir a R' la regla r'_j obtenida tras cambiar en r_j la acción s_i con $s_{i==x}$ (o respectivamente, !=x).

10. Para cada acción s_i, que se ejecute en una regla r_j donde el atributo aparezca en la guarda como $<= x$, $< x$, $> x$, o $>= x$:

    - proceder como en el paso 7 pero añadiendo al conjunto Σ'_3 la acción $s_{i_{comp}}$ (con $comp \in \{<= x, < x, > x, >= x\}$) y, tal que, si teníamos la transición $q_a \xrightarrow{s_i} q_b$ ahora tenemos $q_{a_y} \xrightarrow{s_{i_{comp}}} q_{b_{y+\alpha}}$ para valores válidos de y en cada caso y en cada actualización (sin salirnos de rango),

    - añadir a R' la regla r'_j obtenida tras cambiar en r_j la acción s_i con $s_{i_{comp}}$.

11. Cambiar el autómata original y sus reglas ACE por el nuevo autómata y las nuevas reglas AC.

12. Iterar con el siguiente atributo volviendo al paso 2 o con el siguiente autómata del sistema volviendo al paso 1.

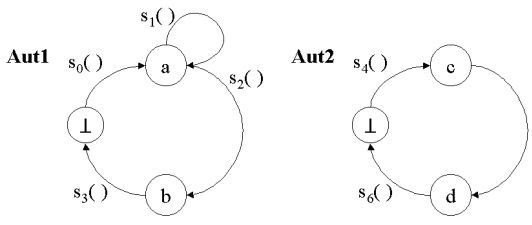

Figura 4. Sistema de Autómatas Cooperativos Extendidos

Veamos un ejemplo de transformación de un sistema genérico y sencillo de Autómatas Cooperativos Extendidos que incluye la mayoría de los casos en otro equivalente de Autómatas Cooperativos. Sean Aut1 y Aut2 los autómatas de la Figura 4 y las reglas siguientes:

name: Rule1
synch: Aut1.$s_0()$ + Aut2.$s_4()$
guard: null
update: Aut1.V=0

name: Rule2
synch: Aut1.$s_1()$
guard: null
update: Aut1.V=Aut1.V+1

name: Rule3
synch: Aut1.$s_2()$ + Aut2.$s_5()$
guard: Aut1.V=2
update: null

name: Rule4
synch: Aut1.$s_2()$ + Aut2.$s_5()$
guard: Aut1.V!=2
update: null

name: Rule5
synch: Aut1.$s_3()$ + Aut2.$s_6()$
guard: null
update: null

Este ejemplo contiene dos autómatas y cinco reglas. El autómata Aut1 contiene un atributo numérico que puede tomar diferentes valores. Por simplicidad, reducimos el ejemplo a que pueda tomar valores entre 0 y 3, ambos incluidos.

El conjunto de estados del autómata incluye tres estados: el estado inicial "dummy", el estado a y el estado b. Así pues, $Q = \{\perp\} \cup \{a, b\}$.

Aplicando el algoritmo sobre el conjunto de estados, obtenemos un nuevo conjunto de estados de un autómata sin atributos numéricos $Q' = \{a_0, a_1, a_2, a_3, b_0, b_1, b_2, b_3\}$.

Realizamos un tratamiento similar con el conjunto de acciones $\Sigma = \{s_0, s_1, s_2, s_3\}$ obteniendo los conjuntos $\Sigma'_1 = \{s'_0, s'_3\}$, $\Sigma'_2 = \{s^1_1\}$ y $\Sigma'_3 = \{s_{2_2}, s_{2_{!2}}\}$.

El autómata resultante puede verse en la Figura 5. Las reglas obtenidas aplicando el algoritmo son las siguientes:

name: Rule1
synch: Aut1'.$s'_0()$ + Aut2.$s_4()$
guard: null
update: null

name: Rule3
synch: Aut1'.$s_{2_2}()$ + Aut2.$s_5()$
guard: null
update: null

name: Rule4
synch: Aut1'.$s_{2_{!2}}()$ + Aut2.$s_5()$
guard: null
update: null

name: Rule5
synch: Aut1'.$s'_3()$ + Aut2.$s_6()$
guard: null
update: null

name: Rule2
synch: Aut1'.$s^1_1()$
guard: null
update: null

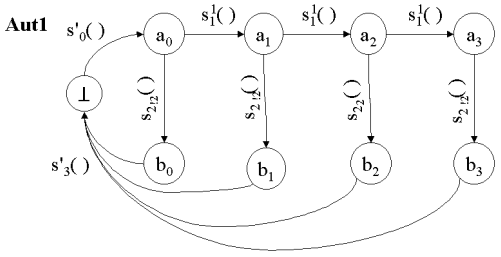

Figura 5. Autómata Cooperativo sin atributos numéricos

En este caso, el número total de reglas es el mismo, pero este número será normalmente mucho mayor, del mismo modo que se incrementa el número de acciones y de estados.

Limitando el rango de valores posibles de los atributos numéricos obtenemos un número finito de estados en el modelo de AC con la misma expresividad del original ACE. El modelo de ACE sólo decrementa la complejidad del modelo resultante en aquellos casos en los que es importante reflejar las posiciones relativas en un autómata sin cambiar la capacidad expresiva con respecto a la versión sin esta extensión. Esta disminución de la complejidad es importante, por tanto, en aquellos sistemas en los que tenemos que comparar posiciones de un autómata con otros o con un valor dado, lo cual es trivial en un sistema ACE pero incrementa exponencialmente la complejidad del modelo AC.

Puede concluirse, por tanto, que el modelo extendido es igual de expresivo que el original de los Autómatas Cooperativos.

Proposición 1 *Un sistema de Autómatas Cooperativos Extendidos puede ser transformado en*

un sistema de Autómatas Cooperativos equivalente.

Podemos, entonces, trasladar las conclusiones acerca de la decibilidad de propiedades obtenidas en el modelo original al modelo extendido, por ejemplo como ocurre con la propiedad de acotamiento.

Corolario 1 *La propiedad de acotamiento de un sistema de Autómatas Cooperativos Extendidos con un solo atributo identificador es decidible.*

5. Conclusiones

En este trabajo se ha presentado el modelo de los Autómatas Cooperativos y se ha puesto de manifiesto la complejidad de los modelos resultantes en aquellos sistemas que, aunque conceptualmente simples, incluyen cambios en el diagrama de estados de algún autómata como consecuencia de su cambio de posición geográfica. Si bien el modelo de los Autómatas Cooperativos es capaz de representar dichos sistemas, la confusión del diagrama de estados resultante es importante. Para lograr una mayor claridad en este tipo de sistemas se ha propuesto una extensión al modelo original: los Autómatas Cooperativos Extendidos. Este nuevo modelo reduce considerablemente el diagrama de estados de cada autómata de estas características sin incrementar la complejidad ni el número de reglas. Esta extensión define un nuevo tipo de atributos (los atributos numéricos) que no representan sino una información adicional al estado del autómata, sin intervenir en la pertenencia a tareas como los atributos originales. Para mostrar que esta extensión conserva las propiedades deseables del modelo original se ha presentado, como último resultado de este trabajo, un algoritmo que permite transformar cualquier sistema ACE en uno AC equivalente. Con este algoritmo, basado en la idea general de que si se limita el rango de valores de los atributos numéricos se puede considerar que forman parte del diagrama de estados, se prueba que un sistema de Autómatas Cooperativos Extendidos es, a su vez, equivalente a un sistema de Autómatas Cooperativos. Además, por definición, todo sistema basado en Autómatas Cooperativos lo es también en el modelo extendido. Con esto, se prueba, entre otras, que la propiedad de acotamiento es decidible en el modelo de los Autómatas Cooperativos Extendidos.

Agradecimientos

Este trabajo ha sido parcialmente financiado por CICYT TIC 2001-2705-C03-01, por la Acción Integrada Hispano-Alemana HA2001-0059, por el Proyecto UPV 7176 y por el Centro de Cooperación al Desarrollo de la UPV.

Referencias

[1] A. Arnold and M. Mivat. Comportements de processus. *Colloque AFCET: Les Mathémátiques de l'Informatique*, pages 35–68, 1982.

[2] E. Badouel, Ph. Darondeau, D. Quichaud, and A. Tokmakoff. Modelling Dynamic Agent Systems with Cooperating Automata. *Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, pages 11–17, 1999.

[3] J.P. Banâtre and D. Le Métayer. Gamma and the Chemical Reaction Model: Fifteen Years After. In Cristian Calude, Gheorghe Paun, Grzegorz Rozenberg, and Arto Salomaa, editors, *WMP*, volume 2235 of *Lecture Notes in Computer Science*, pages 17–44. Springer, 2001.

[4] G. De Michelis and C.A. Ellis. Computer Supported Cooperative Work and Petri Nets. *Lecture Notes in Computer Science*, 1492:125–153, 1998.

[5] C.A. Ellis and J. Wainer. *Groupware and Computer Supported Cooperative Work*, pages 425–457. Gerhard Weiss, 1999.

[6] C.A. Ellis. Team Automata for Groupware Systems. In S. C. Hayne and W. Prinz, editors, *Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work: The Integration Challenge (GROUP'97), Phoenix, AZ, U.S.A.*, pages 415–424. ACM Press, New York, 1997.

[7] S. Greenberg. Computer Supported Cooperative Work and Groupware: An Introduction to the. In *International Journal of Man Machine Studies*, volume 34, pages 133–143. special edition, 1991.

[8] C. Herrero and J. Oliver. Autómatas Cooperativos para Modelizar Sistemas Distribuidos. *Proc. of IX Jornadas de Concurrencia*, pages 115–128, 2001.

[9] C. Herrero and J. Oliver. Extended Cooperating Automata. *Proc. of 2003 IEEE International Conference on Systems, Man & Cybernetics (SMC 2003)*, pages 402–408, 2003.

[10] J.L. Peterson. *Petri Net Theory and the Modeling of Systems.* Englewood Cliffs, New Jersey: Prentice Hall, Inc., 1981.

[11] M.H. ter Beek, C.A. Ellis, K. Kleijn, and G. Rozenberg. Team Automata for CSCW. *Proc. of 2nd Int. Coll. on Petri Net Technologies for Modelling Communication Based Systems*, pages 1–20, 2001.