Un mecanismo de resolución de conflictos en entornos de Inteligencia Ambiental

Pablo A. Haya, Germán Montoro, Xavier Alamán
Dept. de Ingeniería Informática
UAM - EPS
C. Francisco Tomas y Valiente, 11
28049 Madrid
Pablo.Haya@uam.es

Resumen

Los escenarios de *Inteligencia Ambiental* describen situaciones en las que conviven multitud de dispositivos y agentes. Es frecuente en este tipo de escenarios surjan conflictos a la hora de modificar el estado de un dispositivo concreto, como puede ser por ejemplo una lámpara. El problema no es tanto de compartición de recursos como de conflicto de órdenes provenientes de diferentes agentes. Frente a decisiones incompatibles sobre el estado en el que debe ser puesto un dispositivo, hay que tomar una decisión. En este artículo se propone un mecanismo centralizado basado en colas FIFO priorizadas para decidir el orden en que se concede el control del dispositivo. El cálculo de la prioridad de los comandos se establece siguiendo una política que tiene en cuenta el rol de emisor, el tipo de comando, el estado del entorno, y las relaciones entre el emisor y entorno y entre el emisor y el recurso. Finalmente, se proponen una serie de políticas particulares para aquellos recursos que no se ajusten a la política general.

1. Motivación

El término de *Inteligencia Ambiental* (AmI) surge en 1999 a partir de un informe [1] del *Comité de Expertos* del *European Community's Information Society Technology Programme* (ISTAG). En el 2001, el propio ISTAG publica un conjunto de escenarios y recomendaciones que definen la visión de la AmI.

La AmI propone una nueva *Sociedad de la Información* centrada en el usuario. Para ello requiere de servicios eficientes que proporcionen una interacción más amigable. En esta nueva visión las personas se hayan rodeadas de miles de dispositivos e interfaces inteligentes mimetizadas en objetos de la vida cotidiana. A su vez, dispositivos y usuarios se encuentran en un entorno capaz de razonar y de responder de forma personalizada. El objetivo final es conseguir que las tecnologías de la información ayuden de manera no invasiva en el quehacer diario de las personas.

Las raíces de la AmI se hayan en la combinación de tres tecnologías: la *Computación Ubicua* [2], las *aplicaciones sensibles al contexto* [3] y los *Entornos Inteligentes* [4]. Estas tres áreas han sido objeto de estudio en mayor o menor medida dentro del Departamento de Ingeniería Informática de la UAM. Para ello se ha construido un entorno basado en un salón de un hogar digital. Recientemente, el estudio se está trasladando a un entorno educativo.

En un entorno compartido por numerosos componentes distribuidos surgen diversos problemas en el control de acceso a los recursos. Uno de los problemas que aparece con más frecuencia es decidir quién puede acceder a cada uno de los recursos. Dentro del área de los Entornos Inteligentes se han propuesto diferentes soluciones que se engloban en basadas en políticas de seguridad [5] y en listas de control de acceso [6].

Una vez establecido el mecanismo de seguridad hay que afrontar el problema que surge cuando dos o más componentes pretenden controlar de manera simultánea un mismo recurso (por ejemplo un dispositivo, digamos, una lámpara). La solución al conflicto pasa por elegir a qué componente se le concede el permiso para acceder primero al recurso. Un problema relacionado con este ha sido estudiado dentro del dominio de los sistemas distribuidos y se ha denominado exclusión mutua distribuida.

Las soluciones más cercanas a los entornos inteligentes se encuentran en los sistemas multiagente. La *Open Agent Architecture* [7] o

Actas del Simposio de Computación Ubicua e Inteligencia Ambiental, UCAmI2005, pp.11-18
ISBN: 84-9732-442-0

Hive [8] son dos plataformas que han abordado este problema. Otra aproximación es realizar una implementación centralizada [9]. Para ello se requiere de un componente coordinador que se encarga de recibir todas las peticiones y que decide a qué componente otorgarle el permiso para acceder al recurso. Los algoritmos distribuidos aportan frente a la solución centralizada una mayor robustez al no depender de un componente central. Por contra, son menos eficientes en cuanto al número de mensajes que se tienen que transmitir entre los procesos, y adolecen de una mayor complejidad de implementación.

Nuestra propuesta para solventar conflictos dentro de un entorno inteligente aboga por la simplicidad y la eficiencia. Por ello, se ha elegido una solución centralizada. Por otra parte, la capa intermedia que sirve de pegamento entre los componentes del entorno inteligente (véase apartado 2) descansa en un repositorio común de información, lo cual facilita la implementación de un algoritmo centralizado.

En los escenarios que se plantean en un entorno inteligente la priorización entre las peticiones se torna esencial. En general, es frecuente que los usuarios del entorno tengan diferentes roles (padre/hijo, propietario/invitado, profesor/alumno, administrador/usuario…), y que se quiera establecer distinción según la acción la realice uno u otro. De esta forma a un usuario se le da preferencia en el acceso en función de rol que desempeñe. Ahora bien, en un entorno inteligente se plantea otra cuestión a resolver que los mecanismos anteriormente citados no tienen en cuenta. Son frecuentes las situaciones en las que un usuario requiere asumir el control de un recurso que está siendo empleado por otro. Teniendo en cuenta los distintos roles y la situación del entorno la decisión a tomar puede ser la de arrebatar el control al componente que actualmente tiene en su posesión el recurso. Este tipo de decisiones son críticas en escenarios como el hogar; por ejemplo, los módulos de seguridad (gas, inundación, antirrobo…) deben tener preferencia frente al resto de los módulos en caso de emergencia.

Nuestra propuesta consiste en un mecanismo de colas centralizado en el que se prioricen las acciones que se realicen sobre los recursos. Las peticiones se almacenan en la cola teniendo en cuenta la prioridad, de forma que la que se extrae primero es la que más prioridad tiene. Finalmente, se ha decidido emplear colas de prioridades preferentes. Estas colas se caracterizan porque si la petición que llega tiene más preferencia que la que está en curso, le arrebata el control del recurso. En contraposición, en las colas no preferentes la petición que llega tiene siempre que esperar a que termine la petición en curso. Ahora bien, este mecanismo obliga a los agentes tener en cuenta que su petición puede llegar a no ser atendida. Esto es así porque las peticiones de baja prioridad pueden permanecer indefinidamente en la cola, si llegan continuamente peticiones de mayor prioridad.

Así, hay que contemplar el caso en el que un componente con elevada prioridad se adueñe de forma indefinida del recurso. Un método simple y efectivo de solventar la espera indefinida es establecer un tiempo límite para la reserva de un recurso. Por ejemplo, en la capa intermedia Jini [10] se plantea un mecanismo que consiste en permitir a un componente reservar el acceso de un recurso durante un tiempo fijo. Para seguir manteniendo el control del recurso el componente tiene que renovar la concesión antes de que caduque.

De forma similar, para evitar que los procesos se queden perpetuamente esperando a que se cumpla la petición, se incorpora un tiempo de caducidad a las peticiones de forma que pasado este tiempo dejen de ser válidas. Este límite temporal es establecido por el proceso que emite la petición, de forma que puede controlar si ha caducado o no.

En el siguiente apartado se va a resumir la arquitectura de la capa intermedia implementada. En los posteriores apartados se desarrollará la propuesta e implementación del mecanismo de resolución de conflictos, y cómo se integra dentro de la capa intermedia.

2. Capa de contexto

En el Departamento de Ingeniería Informática de la UAM se ha habilitado un entorno domótico simulando un salón de un hogar convencional [11] Este entorno posee diversos componentes hardware y software que permiten controlar los diferentes dispositivos del salón. Se han desarrollado varias aplicaciones sensibles al contexto del usuario y se ha experimentado con interfaces de usuario a través de la web [12] y con diálogo natural [13].

Para facilitar la integración de dispositivos, aplicaciones e interfaces se ha desarrollado una

capa intermedia. Esta capa intermedia se ha denominado capa de contexto, ya que el mecanismo de coordinación se centra en el intercambio de contexto entre los componentes del entorno.

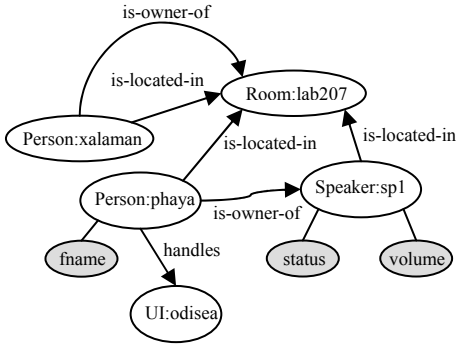

Figura 1. La pizarra almacena la información sobre el entorno *lab207*. Este *contiene* dos personas (*xalaman* y *phaya*) y un altavoz. (sp1). *xalaman* es el *propietario* del entorno, y *phaya* es el *propietario* del altavoz, y está *empleando* la interfaz de usuario *odisea*

La capa contexto propuesta está basada en una arquitectura de tipo pizarra cuyas características son:

- *Un modelo de datos común*. La información que se almacena en la pizarra sigue un modelo común. El contexto del entorno se representa mediante entidades definidas por un conjunto de propiedades. A su vez se puede establecer relaciones entre las entidades formando un grafo.
- *Un repositorio central*. La pizarra almacena toda la información contextual del entorno. La información que se guarda puede ser un cambio en una propiedad (una puerta que se abre), o una nueva entidad que se añade o se elimina de la pizarra (alguien qué entra o sale de la habitación).
- *Un mecanismo de eventos asíncrono*. Además de acceder directamente a la información, las fuentes de información publican los cambios en el contexto. Los consumidores que se encuentran subscritos reciben estos cambios de la pizarra.

El contexto proviene de fuentes información de distinta naturaleza. Esta se almacena en la pizarra en forma de un grafo de entidades, donde cada entidad se representa siguiendo un esquema común. Así, los agentes manejan una visión unificada del contexto, independientemente de su origen y de su nivel de abstracción. En la Figura 1 se observa un ejemplo simplificado de una instantánea de la pizarra.

Finalmente, la capa de contexto provee de un conjunto de operaciones para gestionar el contexto. Estas son:

- *Obtener contexto*. Permite recuperar la definición de una entidad, el valor de una propiedad o la relación que guarda una entidad con otras determinadas.
- *Modificar propiedad*. Cambia el valor de un atributo de una entidad. Si la entidad tiene un reflejo directo en el entorno, se modifica éste en consecuencia.
- *Añadir entidad*. Añade una nueva entidad a la pizarra reflejando la aparición de un nuevo componente o la generación de nueva información contextual.
- *Borrar entidad*. Elimina una entidad que ha dejado de pertenecer al entorno.
- *Añadir relación*. Establece una relación entre dos entidades.
- *Borrar relación*. Borra una relación que hubiera entre dos entidades.
- *Subscribirse*. Permite subscribirse a cambios en la pizarra, por ejemplo un cambio de valor en una propiedad, una entidad que entra o sale, o una relación que aparece o desaparece.
- *Borrar suscripción*. Elimina una suscripción.

3. Un mecanismo basado en colas de prioridades

Cuando un agente pretende realizar una operación, envía un comando que es recibido por la capa de contexto. Las operaciones de *Modificar propiedad* y de *Añadir/Borrar relación* cambian el estado de las entidades y la configuración del entorno, por lo que son susceptibles que se produzcan conflictos si varios agentes envían una operación que afecta a la misma propiedad o relación. Esto es particularmente problemático para acciones que implican cambios físicos en el entorno, como puede ser el encender o apagar dispositivos, cambiar el canal de la televisión o el volumen del altavoz.

Para solventar este tipo de situaciones, se provee de una cola por cada propiedad de una entidad y por cada relación entre dos entidades. Cuando se envía un comando de los anteriormente descritos se le asigna una prioridad (véase el

apartado 4), además de otra serie de parámetros (véase el apartado 3.1). A continuación el comando se almacena en la cola correspondiente. La cola de prioridades a la cual se envía un comando se determina en función de la propiedad o relación sobre la cual se quiere actuar.

En cada instante de tiempo, cada cola se compone de un conjunto de comandos ordenados según su prioridad, entre los cuales aquel que tiene mayor prioridad se denomina comando activo. Cuando llega un nuevo comando se almacena en la cola de forma que si su prioridad es mayor que la prioridad del comando activo, el recién llegado pasa a ser el comando activo. En caso contrario, se guardará en el lugar correspondiente de la cola. Siempre que un comando se activa, este se ejecuta una vez y se aplican los cambios pertinentes en la pizarra (y en el entorno físico en su caso).

Para evitar el problema de que un comando con la máxima prioridad bloquee la cola indefinidamente se establece un tiempo de caducidad para cada comando. De esta manera, una vez se ha ejecutado el comando permanecerá como comando activo hasta que caduque o hasta que otro comando con mayor prioridad le reemplace. Por ejemplo, una orden directa por parte de un usuario para encender la luz de una habitación tendrá prioridad sobre otros comandos enviados por el módulo de ahorro de energía, pero sólo durante un intervalo de tiempo previamente prefijado. Un comando se ejecutará tanta veces como se active, es decir, tantas veces como acceda a la primera posición de la cola y no haya caducado.

Una restricción que se impone es que para cada cola sólo puede existir un comando por cada agente. Si un agente envía un comando a una cola, se elimina previamente el comando anterior, en el caso de que existiera. Por otro lado, los agentes tienen la capacidad de anular un comando de una cola determinada, o todos los comandos enviados a todas las colas.

Las colas se crean y se destruyen según las necesidades. Cuando llega un comando se comprueba si ya existe la cola correspondiente, y en caso contrario se crea. Cuando una cola se queda vacía de comandos, se elimina. De esta forma podrá haber tantas colas de prioridades como propiedades y relaciones haya en la pizarra, pero teniendo en cuenta que en memoria únicamente estarán aquellas colas que tengan una o más ordenes.

En la Figura 2 se muestra el comportamiento de una cola según llegan diferentes comandos con diferentes prioridades y tiempo de caducidad.

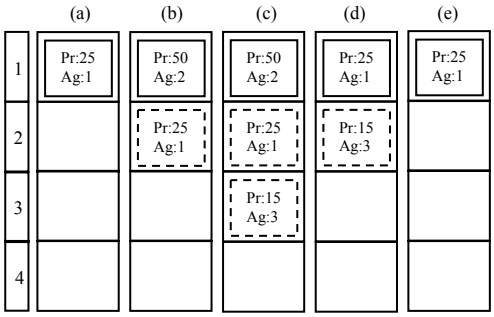

Figura 2. Evolución de una cola de prioridades cuando llegan ordenes de tres agentes (Ag 1, Ag 2 y Ag 3) con distintas prioridades (Pr)

3.1. Parámetros de un comando

Aparte de la prioridad, se establecen una serie de parámetros para cada comando. Estos son los siguientes:

- *Propietario del comando*. Identifica qué agente ha enviado el comando.
- *Prioridad*. Un entero positivo. Cuanto mayor sea la prioridad antes se ejecutará el comando.
- *Modo*. Determina cómo se calcula caducidad del comando (ver Tabla 1).
- *Tiempo de caducidad*. Indica a partir de qué fecha y hora el comando deja de tener validez. Este parámetro sólo tiene validez si el modo elegido es "fecha de caducidad". La fecha se mide en milisegundos.
- *Nombre*. Asocia un nombre al comando. Dado un propietario, este parámetro permite distinguir entre los comandos emitidos. Este nombre permite al agente referenciar al comando para eliminarlo de la cola. Varios comandos se pueden agrupar bajo el mismo nombre, de tal forma que se pueden borrar todos a la vez.

El *propietario del comando*, *modo*, *tiempo de caducidad* y *nombre* son establecidos por el agente que envía el comando, mientras que la *prioridad* la decide la capa de contexto en función de la política de prioridades (véase apartado 4.1).

La Tabla 1 describe los distintos modos en que el emisor puede elegir la caducidad de un comando.

MODO	DESCRIPCIÓN
Tiempo de caducidad	La caducidad de la orden se establece mediante el tiempo en milisegundos que indique el parámetro *Tiempo de caducidad*.
Después de ejecutarse	El comando desaparece de la cola de prioridad una vez que se haya ejecutado. Por defecto se establece un tiempo de caducidad máximo de modo que si el comando no se ejecutó en ese tiempo se elimina
Instantánea	La orden no se almacena en la cola, si no existe ninguna orden con mayor prioridad en la cola, se ejecuta y desaparece.

Tabla 1. Distintos modos de establecer la caducidad de un comando

4. Políticas para establecer las prioridades de los comandos

El mecanismo de prioridades decide el orden en que se ejecutaran un conjunto de comandos que actúan sobre un recurso. Este orden se establece en función de la prioridad del comando y del tiempo de validez de esta prioridad. Sin embargo, no se ha definido en el mecanismo anterior cómo asignar la prioridad a un comando. Para ello se dispone de un módulo que se encarga de asignar las prioridades de los comandos. Cada política es una función que decide, dado un comando, cuál es la prioridad que se le debe asignar. Se podrían definir tantas políticas como recursos se quieran controlar aunque, por motivos prácticos, por defecto todos los comandos se tratan siguiendo una misma política, y aquellos casos particulares que no se ajustan a la política por defecto se tratan por separado mediante políticas específicas.

El administrador del entorno es el encargado de decidir para cada recurso cuál es la política que se quiere aplicar.

4.1. Política de asignación de prioridades

Este apartado describe la política que se emplea por defecto. El diseño de esta política viene condicionado por dos consideraciones. Por un lado, la política tiene que ser lo suficientemente flexible y genérica para abarcar el mayor número posible de casos. Por otro, la implementación tiene que ser sencilla para que no suponga una sobrecarga computacional excesiva.

Para sopesar la prioridad del comando se tienen en cuenta distintas variables. Estas se pueden dividir en tres grupos: (1) Aquellas relativas al emisor de la orden; (2) Aquellas relativas al comando; y (3) Aquellas relativas al entorno.

Variables relativas al emisor de la orden

Dentro del primer grupo se incluyen quién es el *emisor de la orden* y el *rol* con el cual está actuando. La primera variable puede tener dos valores: o bien que el comando provenga de un *usuario* (U) o bien de una *aplicación* (A). Esta variable indica realmente quién es el que ha decidido emitir la orden, ya que los comandos provenientes de los usuarios siempre son a través de algún tipo de interfaz, ya sea un componente software (una interfaz gráfica, textual, vocal...) o un componente físico (un interruptor, un panel de control, un regulador...). En contraposición, se entiende que el emisor es una aplicación cuando la decisión del comando la toma una aplicación sin intervención directa del usuario.

Por otro lado, el *emisor del comando*, independientemente de que sea un usuario o una aplicación, puede asumir distintos *roles* dependiendo de la relación entre él y el recurso que se quiere modificar. El emisor puede ser *propietario* (O) del recurso, o puede ser *invitado* (G). Ambos roles se indican según exista o no una relación en la pizarra del tipo *propietario-de* entre las entidades que representan al emisor y al recurso. Además, la relación *propietario-de* también se puede establecer entre un emisor y un entorno. En este caso, el emisor se considera propietario de todos los recursos que se encuentren dentro de éste.

Variables relativas al comando

El segundo grupo está formado por una única variable qué indica el *tipo de orden*. Esta puede tomar distintos valores según quién emita el comando. Si el emisor es un usuario, el tipo de orden puede ser un *comando directo* (C) o una *preferencia de usuario* (P). En el primer caso, la orden se ha generado como resultado de una acción directa del usuario, como por ejemplo, pulsar un botón. En el segundo caso, la orden se

origina automáticamente tras cumplirse una condición que el usuario previamente había impuesto. En el caso de que el emisor sea una aplicación, se definen dos tipos de comandos según que correspondan a: (a) *comandos ordinarios* (L) y (b) *comandos de seguridad* (H). Estos últimos son aquellos comandos que provienen de aplicaciones que se encargan de la seguridad del entorno, mientras que los primeros son el resto.

Variables relativas al entorno

Finalmente, el último grupo engloba dos variables relativas al estado del entorno. La primera define el *nivel de alerta*, que puede ser *normal* o de *emergencia*. Este segundo valor se reserva para situaciones graves como, por ejemplo, en caso de incendio o inundación. La segunda define el *nivel de seguridad*. Se definen tres niveles de seguridad: *bajo*, *normal* y *alto*. Una habitación con nivel de seguridad bajo correspondería a una localización pública y compartida por varios usuarios, un nivel normal se podría establecer para localizaciones privadas y pertenecientes a un usuario, y un nivel alto, para entornos que tuvieran restricciones de seguridad importantes.

4.2. Cálculo de la prioridad

Cuando se recibe un comando, se recupera el valor de cada una de las variables anteriores. Cada posible emisor tiene un código único que le autentifica y que se envía junto con la orden. Este código permite averiguar a partir de la definición del emisor en la pizarra quién es el emisor de la orden y qué tipo de órdenes emite. Mediante la relación del emisor y el recurso que se quiere modificar se halla el rol de emisor. Finalmente, se obtienen el estado de alerta y de seguridad del entorno a partir de las propiedades de la entidad entorno.

A continuación se forma una tupla con las variables correspondientes a los grupos (a) y (b), de tal forma que el primer elemento defina el emisor, el segundo el rol, y el tercero el tipo de orden. Por ejemplo, un usuario propietario del recurso que ha enviado una orden directa se define mediante la tupla (UOC), mientras que una aplicación invitada sería la tupla (AGL).

La prioridad se calculará según la posición que ocupa la tupla en una lista. El orden de la lista será en función del estado de alerta del entorno y el nivel de seguridad. Tal como se muestra en la Tabla 2, se han definido cuatro listas donde se ordenan las tuplas de mayor a menor prioridad.

	Orden de preferencias			
	Emerg.	Seguridad		
		Bajo	Normal	Alto
1	UOC	UOC	UOC	AOH
2	UGC	UGC	AOH	UOC
3	AOH	UOP	UGC	AOL
4	AGH	AOH	UOP	UOP
5	AOL	AOL	AOL	UGC
6	AGL	UGP	UGP	UGP
7	UOP	AGH	AGH	AGH
8	UGP	AGL	AGL	AGL

Tabla 2. Listas correspondientes a los estados de alerta y seguridad ordenadas según las prioridades de las tuplas

Una vez se recupera la posición que ocupa la tupla en la lista correspondiente, se calcula la prioridad según la ecuación 1.

$$Prio = (MAXPRIO - MINPRIO) / pos \qquad (1)$$

Donde MAXPRIO y MINPRIO son dos constantes que determinan la máxima y mínima prioridad que se pueden asignar, y *pos* es la posición que ocupa en la lista.

La lista de emergencia se utiliza cuando el entorno se encuentra en estado de emergencia, independientemente del nivel de seguridad. En caso de que el estado sea normal, la lista que se emplea se determina por el nivel de seguridad.

Tal como se comprueba en la lista de emergencia, priman las ordenes directas de los usuarios, ya sea propietarios o no. A continuación se tendrán en cuenta las ordenes de las aplicaciones propietarias del entorno, seguidos de aquellas que son invitados. Finalmente, las órdenes con menor prioridad serán las preferencias de los usuarios.

Para una situación normal del entorno y un nivel de seguridad bajo, las acciones de los usuarios, tanto propietarios como invitados, tendrán la máxima prioridad. A continuación, las preferencias de los usuario propietarios, y los comandos enviados por las aplicaciones pertenecientes al entorno. Finalmente, se aplicarán las preferencias de los invitados y las aplicaciones externas al entorno.

En el caso de un nivel de seguridad normal, la lista de prioridades se mantiene idéntica a la anterior excepto que los módulos de seguridad alcanzan la segunda posición en la lista, únicamente superados por las acciones directas de los propietarios.

Por último, la lista de nivel de seguridad alto pone como comandos más prioritarios aquellos enviados por módulos de seguridad del entorno, seguido por los comandos emitidos por usuarios propietarios o resto de aplicaciones del entorno. Las acciones de usuarios o aplicaciones que no tienen relación con el entorno se les otorga menor prioridad.

Este mecanismo permite realizar cambios fácilmente, y adaptar rápidamente la política general de prioridades a nuevas necesidades simplemente cambiando el orden de las listas.

5. Políticas de asignación de prioridades particulares

Pueden existir propiedades de recursos que no se adapten al mecanismo de asignación anteriormente descrito. Por ejemplo porque la prioridad dependiera del valor de la propiedad, lo cual no está contemplado en la política anterior. Así, los cambios en el volumen de unos altavoces localizados en el entorno tienen que seguir unas convenciones sociales que no están recogidas en la política genérica. En este caso, tiene más sentido emplear una política específica para decidir cuál de los posibles valores es el que se establece para el volumen del altavoz. Si dos o más agentes quieren subir o bajar el volumen, las normas sociales, por lo general, determinan que se ponga el volumen más bajo. En especial, si se trata de preferencias de usuario que se disparan automáticamente. Para poder reflejar este tipo de comportamiento es necesario añadir un mecanismo que permite asociar políticas definidas *ad-hoc* que sustituyan a la política por defecto en determinadas propiedades.

```
<class name="speaker">
  <property name="Left_Volume">
    <paramSet name="policy">
      <param name="ad-hoc">
          polite_speaker_volume
      </param>
    </paramSet>
  <property name="Right_Volume">
    <paramSet name="policy">
      <param name="ad-hoc">
          polite_speaker_volume
      </param>
    </paramSet>
  </property>
</class>
```

Figura 3. Definición de la política de asignación de prioridades de una entidad altavoz empleando XML

En el ejemplo de la Figura 3 se ha empleado un lenguaje de definición de la pizarra [14] basado en XML. Este lenguaje permite asociar a todas las entidades de una misma clase una política particular. Para ello se introduce en la definición de la clase un parámetro denominado *ad-hoc*. El valor de este parámetro será una cadena de texto con el nombre de la política especial que se aplicará a esa propiedad.

Esta política podrá ser escogida dentro de una de las tres políticas que se describen más abajo. Este conjunto se puede ampliar fácilmente a medida que se necesitae nuevas políticas.

Actualmente se incluyen tres políticas particulares:

- *Volumen altavoz*. La prioridad es inversamente proporcional al volumen qué se quiera establecer en altavoz.
- *Primar apagar/cerrar*. Si la orden es de apagar (o cerrar) el recurso, está tiene la máxima prioridad. Sería también el caso del altavoz. Si un emisor decide apagarlo completamente tiene más prioridad que los que deciden mantenerlo encendido.
- *Primar encender/abrir*. Este sería el caso contrario al anterior. Se prima a aquellos comandos que enciendan (o abran) el recurso. Por ejemplo, sería la política a aplicar para el control de una puerta.

Hay que tener en cuenta que en todas las políticas se trata el estado de emergencia de forma específica.

6. Conclusiones y Trabajo Futuro

En este trabajo se ha presentado una propuesta para conseguir un comportamiento armonioso entre los componentes de un entorno de *Inteligencia Ambiental*. Para ello se propone un mecanismo centralizado de asignación de prioridades que permite resolver los conflictos entre dos o más agentes que intentan modificar la información gestionada por una arquitectura de pizarra. Esta pizarra contiene toda la información relativa al contexto y componentes del entorno. Las modificaciones que se producen en la pizarra se traducen a los dispositivos y aplicaciones mediante un mecanismo de eventos asíncronos. Estas modificaciones pueden ser en una propiedad de una entidad, o en una relación entre entidades.

Cuando un agente envía una operación esta se almacena en una cola asociada. El mecanismo propuesto decide cuál de los comandos se lleva a cabo en primer lugar según:

- Una política de asignación de prioridades que depende de quién es el emisor de la orden; del tipo de orden; de las relaciones entre el emisor y el recurso, y entre el emisor y el entorno; y del estado en que se encuentra el entorno.
- Una política particular que se define para cada recurso que no se ajuste a la política general.

Mientras que la prioridad se asigna automáticamente, los agentes deciden la caducidad de los comandos, para evitar que estos queden indefinidamente esperando en la cola.

Actualmente se está estudiando si conviene modificar el cálculo de prioridad para que contemple otros aspectos. Por un lado, se podría añadir el tiempo que un comando ha permanecido en la cola, aumentando la prioridad cuanto más tiempo pase. De este modo se garantizaría la propiedad de pervivencia de la exclusión mutua. Por otro lado, se podría incorporar el número de veces que un agente envía un comando, disminuyendo la prioridad de cada sucesiva petición. De esta forma, se evitaría que agentes con una prioridad a priori elevada pudieran acaparar un recurso enviando un gran número de peticiones consecutivas.

Agradecimientos

Este proyecto está financiado por el Ministerio de Ciencia y Tecnología con código TIN2004-03140.

Referencias

[1] Weyrich, C. Orientations for Workprogramme 2000 and beyond. ISTAG Report, 1999

[2] Weiser, M. Some computer science issues in Ubiquitous Computing. Communications of ACM, 36(7) pp. 75-84, 1993.

[3] Schilit, B. et al. Context-Aware Computing Applications, IEEE Workshop on Mobile Computing Systems and Applications, 1994.

[4] Cook, D. and Das, S. Smart Environments Technologies, Protocols and Applications, Wiley-Interscience, 2005.

[5] Kagal, L. et al. A Policy Language for a Pervasive Computing Environment. IEEE 4th Int. Workshop on Policies for Distributed Systems and Networks. 2004.

[6] Rattapoom T. Security and Privacy in the Intelligent Room, Master's Thesis for MIT, 2002.

[7] Martin, D. et al. A framework for building distributed software systems, 13(1&2), pp. 91-128, 1999.

[8] Minar, N. et al. Hive: Distributed agents for networking things. In Proceedings of ASA/MA'99, 1999.

[9] Gajos, K. Rascal - a Resource Manager For Multi Agent Systems In Smart Spaces, CEEMAS, 2001

[10] Arnold, K. et al. The Jini Specification, Addison-Wesley, 1999.

[11] Haya, P. et al. A prototype of a context-based architecture for intelligent home environments, CoopIS, 2004.

[12] Alamán, X. et al. Using context information to generate dynamic user interfaces, HCI International, 2003.

[13] Montoro, G. et al. Spoken interaction in intelligent environments: a working system, Advances in Pervasive Computing, Eds. Austrian Computer Society (OCG), 2004.

[14] Haya, P. et al. Extending an XML environment definition language for spoken dialogue and web-based interfaces, AVI, 2004.