UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

PROYECTO FIN DE CARRERA

PERSONALIZACIÓN DE ENTORNOS INTELIGENTES POR EL USUARIO FINAL

Peter Trullenque Eriksson

Abril 2013

PERSONALIZACIÓN DE ENTORNOS INTELIGENTES POR EL USUARIO FINAL

AUTOR: Peter Trullenque Eriksson
TUTOR: Pablo A. Haya Coll

Escuela Politécnica Superior
Universidad Autónoma de Madrid
Abril de 2013

Agradecimientos

A mis padres, por su constante atención y comprensión.
A mi hermana, por las incontables horas pasadas en el salón de su casa después de volver de la universidad, siempre allí cuando lo necesitaba.
A mi tutor, Pablo, por su paciencia, ayudándome cuando me quedaba parado.
A David, por los momentos pasados en la carrera y las oportunidades ofrecidas.
A Ángel, por su apoyo a través de innumerables prácticas y sin quién la realización de este proyecto no habría sido lo mismo.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

1 Introducción

Los entornos inteligentes en el ámbito de la domótica, como se conocen a día de hoy, se remontan a la década de los 90, momento en el cual se empezaron a describir como la combinación de sistemas informáticos y telemáticos para apoyar cualquier actividad diaria del hogar.

Sin embargo, a pesar del interés creado por estos sistemas a finales de dicha década, no hubo el crecimiento esperado debido a la ausencia de un protocolo unificado y los costes de entrada que conllevaba, lo que lo convirtió en un nicho para aficionados y personas con recursos.

Actualmente, como consecuencia del auge de los *smartphones* y *tablets*, se ha producido otra ventana de oportunidad para conseguir acercar esta tecnología a un público más extenso.

Con este público extenso en mente, la pregunta común en trabajos sobre entornos inteligentes de cómo adaptar dichos entornos a las preferencias del usuario pasa a ser inadecuada, remplazada por cómo puede el usuario adaptar el entorno a sus preferencias, como cuestión más interesante.

El usuario final, cuando utiliza un entorno de escritorio, está habituado al uso de diversas aplicaciones ordenadas y generalmente creadas con un propósito específico, y son usadas para tal. Esto mismo ocurre en plataformas móviles como *smartphones* y *tablets*. En cambio, los entornos reales están compuestos por múltiples elementos sin relación necesaria que son operados y administrados por distintas personas con sus preferencias, necesidades y fines, y cuyos objetos pueden ser usados de distintas formas.

Para permitir al usuario final controlar el entorno que le rodea, es necesario que se le permita no solo un control directo de las funciones de los elementos, sino también, un control indirecto, permitiendo definir una guía al sistema para que éste pueda controlar de la forma adecuada la respuesta a los escenarios que ocurran, produciéndose así la personalización de dicho entorno por el usuario.

1.1 Motivación

La personalización de un entorno inteligente es una tarea compleja, ya que implica combinar tecnologías muy diferentes sobre las cuales el usuario final tiene escaso o nulo conocimiento. Históricamente, esta tarea ha quedado reservada a técnicos especializados encargados de programar el entorno según las necesidades requeridas por el usuario.

El problema se produce cuando es necesario realizar una modificación o bien en la configuración inicial debido a cambios en las preferencias del usuario o debido a la introducción de nuevos dispositivos al entorno. Las soluciones que existen actualmente en el mercado se basan en permitir al usuario final el seleccionar entre un conjunto de comportamientos predefinidos.

Estas soluciones son inadecuadas para la personalización de entornos personales como pueden ser el hogar ya que forman parte de la definición individual y de grupo de las personas [1]: en lugar de usarse los elementos que los componen para fines específicos, los usuarios aportan su personalidad y la forma en la que viven sus vidas a la transformación del entorno. Por ello, permitir al usuario retener el control que tienen inicialmente sobre el hogar al pasar a ser un entorno inteligente debe ser una prioridad.

Este proyecto tiene, por tanto, como finalidad la creación de una herramienta software que sea manejada por un usuario final y que facilite la personalización de un entorno inteligente, reduciendo la necesidad del personal técnico.

1.2 Objetivos

El objetivo principal consiste en desarrollar la herramienta mencionada, permitiendo al usuario final definir comportamientos sobre los elementos de un entorno inteligente. Estos comportamientos se ejecutarán cuando se produzcan determinados cambios definidos por el usuario. En particular, se facilitará la creación de comportamientos disparados por cambios en la localización de los elementos (objetos o personas).

Adicionalmente, se desea que la aplicación en cuestión haga uso de un sistema de reglas integrado ya en el entorno existente, de manera que permita al cliente crear y añadir nuevas reglas en función de la localización absoluta o relativa de dichos elementos en el área de acción de las distintas entidades del entorno.

Como requisitos, dicha aplicación debe permitir al usuario definir las reglas de comportamiento de una manera intuitiva, ofreciendo una interfaz gráfica representativa generada por el contexto, con información sobre los elementos del entorno, sus áreas de acción y las operaciones que se pueden llevar a cabo sobre ellos.

Del mismo modo debe contener información sobre la sintaxis del sistema de reglas empleado así como la arquitectura del entorno, dando al usuario la capacidad de construir y administrar conjuntos de reglas de manera sencilla.

Por otro lado, el entorno sobre el cual ha de funcionar la aplicación debe poseer una infraestructura con los dispositivos necesarios para llevar a cabo la localización de los elementos del entorno, siendo toda la información de contexto generada accesible por la aplicación.

1.3 Solución propuesta

Como solución, se propone el uso de una interfaz de usuario web que permita administrar conjuntos de reglas de una forma sencilla e intuitiva, permitiendo crear dichas reglas por medio de módulos anexos a la herramienta principal que facilitan el proceso de manera significativa.

Para implementar dicha solución se hace uso de la arquitectura del entorno, denominada pizarra o *blackboard*, que contiene la información sobre todos los elementos que componen el entorno, así como permitir la creación de agentes, independientes entre sí, encargados de la comprobación y ejecución del conjunto de reglas asociados a cada uno.

Las reglas de comportamiento empleadas siguen el formato ECA (evento-condición-acción), explicado en detalle más adelante, que buscan facilitar la comprensión por parte del usuario final.

Por un lado, la herramienta principal permite administrar los mencionados agentes, simplificando su función a vista del usuario como conjuntos de reglas organizados por éste. Por otro, las herramientas creadas para el diseño de reglas pueden enfocarse a casos particulares, facilitando la interfaz de usuario en cada caso de una forma específica, como puede ser la vista gráfica de localizaciones en el caso ejemplo.

1.4 Organización de la memoria

La memoria consta de los siguientes capítulos:

- **Introducción.** Primer apartado que introduce el problema a solucionar y el acercamiento que el proyecto da al mismo.
- **Trabajo relacionado.** Apartado que describe proyectos similares y su influencia en el proyecto actual.
- **Programando comportamientos con AmIRules.** Descripción del sistema de reglas que emplea el entorno inteligente sobre el cual se basa el proyecto.
- **BBb: Arquitectura software de la herramienta.** Descripción del funcionamiento interno de la herramienta desarrollada.
- **BBb: Interfaz gráfica de usuario.** Descripción del funcionamiento de la herramienta (y su módulo de ejemplo) desde la perspectiva del usuario final.
- **Conclusiones y trabajo futuro.** Conclusiones sacadas del trabajo realizado así como posibles ampliaciones a desarrollar sobre el proyecto finalizado.

2 Trabajo relacionado

Este apartado se centrará, en primer lugar, en entornos inteligentes desarrollados bajo el objetivo de facilitar la personalización final, así como la adaptación a nuevos elementos. Se mostrarán diferentes acercamientos a la idea propuesta, seguido del entorno sobre el cual se ha implementado este proyecto.

Por otro lado, se presentarán proyectos centrados en la creación de herramientas para la personalización de entornos, orientados a la simplificación de la programación de comportamientos.

2.1 Entornos inteligentes

2.1.1 Matilda's Smart House

Figura 2-1: Matilda's Smart House

Este entorno, desarrollado por la Universidad de Florida, consiste en la creación de un laboratorio en el cual se simula un apartamento, que tiene como objetivo realizar pruebas con tecnologías centradas en ayudas a personas mayores o con algún tipo de discapacidad [2].

Entre otros proyectos, se explora el uso de *smartphones* para crear mandos remotos universales permitiendo interactuar a los usuarios con el entorno que los rodea de manera sencilla. Para hacer esto posible, la casa está equipada con múltiples dispositivos: sensores de localización por ultrasonidos, dispositivos X-10 y electrodomésticos aumentados [3].

2.1.2 The Gator Tech Smart House (GTSH)

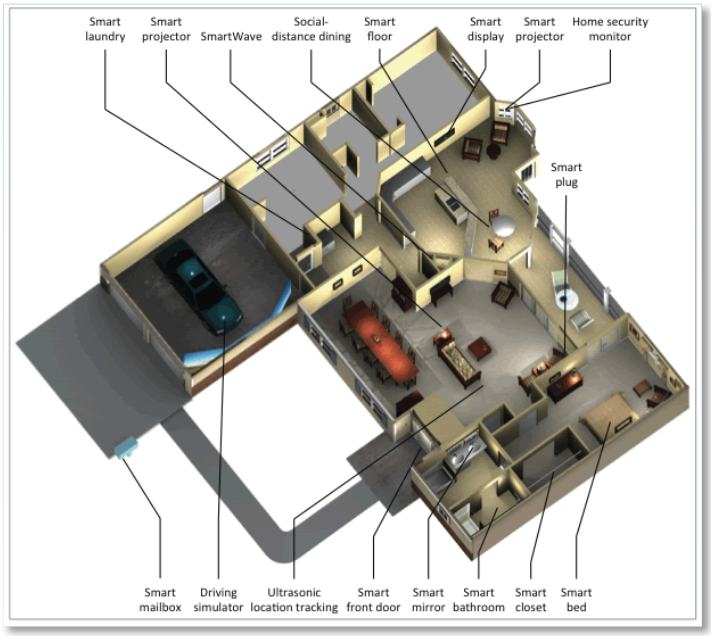

Figura 2-2: Gator Tech Smart House

Basado en el entorno anterior, la Universidad de Florida creó el **Gator Tech Smart House** como continuación en su investigación sobre espacios programables. El objetivo de dicha investigación es diseñar una arquitectura y un *middleware* que permita desarrollar entornos inteligentes de una manera fácil. Se pretende lograr dicho fin creando un IDE que facilite la labor de los diseñadores y los programadores.

A diferencia de **Matilda's Smart House**, este entorno descarta la idea de simulación a favor de una localización real, en este caso, una casa-laboratorio centrada en la investigación a la asistencia a ancianos, maximizando su independencia y calidad de vida.

Para lograr el mencionado objetivo, el apartamento dispone de una plataforma de sensores flexible, debiendo ser identificados automáticamente sin necesidad de configuración [4][5].

2.1.3 AMIGO

Este proyecto, más que enfocarse en la creación de un entorno inteligente, busca integrar dispositivos de dominios diferentes en un mismo sistema central.

Dicho sistema permite lograr un acceso simple al contenido a través del hogar, prediciendo las acciones y/o necesidades del usuario por el contexto asociado. La interfaz del usuario es sencilla y robusta, poniendo un especial énfasis en la privacidad y seguridad.

Por otro lado, se desea una interoperabilidad entre todos los dispositivos, aún de fabricantes diferentes, así como la identificación y actualización automática de dichos dispositivos y servicios [6].

2.1.4 Proyecto SMARTLAB

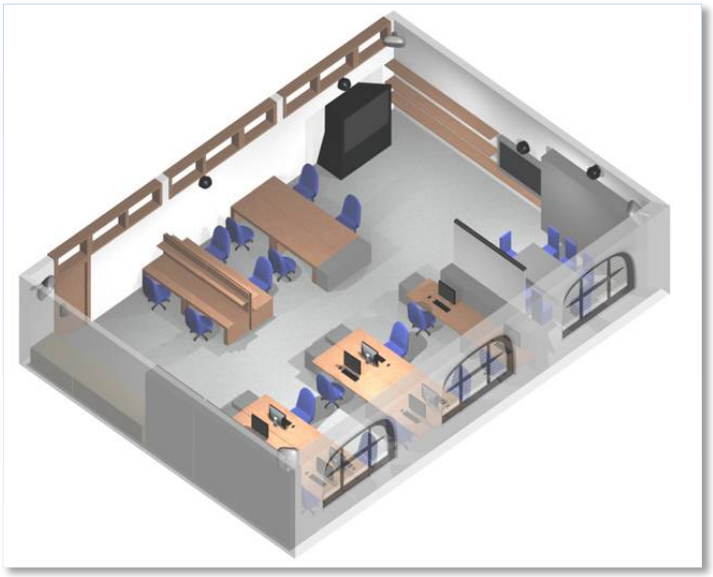

Figura 2-3: Proyecto SMARTLAB

Este proyecto, financiado por el programa del Departamento de Industria, Comercio y Turismo del Gobierno Vasco, tiene como principal objetivo proporcionar una plataforma que permita integrar la sensorización, razonamiento y actuación sobre dispositivos y equipamiento heterogéneos en un entorno inteligente, independientemente de su dominio de aplicación.

Con **SmartLab**, se quiere simplificar la instrumentación de entornos y la exportación de los servicios ofrecidos, así como permitir su evolución al aparecer nuevas tecnologías. En esencia, se pretende superar los entornos inteligentes de primera generación, donde la integración de nuevos elementos heterogéneos era un proceso manual y específico y así, poder crear entornos inteligentes de segunda generación programables y escalables, eficientes en coste, evolutivos y siguiendo estándares definidos [7].

2.1.5 Georgia Aware Home

Figura 2-4: Georgia Aware Home

Creado a través de la iniciativa AHRI (*Aware Home Research Initiative*), programa de investigación llevado a cabo por el *Georgia Institute of Technology*, tiene por objetivo el desarrollo de las tecnologías necesarias para crear un entorno del hogar que pueda percibir y asistir de manera adecuada a sus ocupantes. El laboratorio consta de una casa de tres plantas que sirve de plataforma para diversas pruebas en proyectos centrados en las tecnologías domésticas futuras.

La investigación se centra principalmente en tres áreas: salud y bienestar, que busca ayudar a personas en necesidad de asistencia, áreas de entretenimiento digital, con intención de integrar las nuevas tecnologías en el ámbito del hogar y, por último, sostenibilidad, desarrollando la automatización del control de recursos (gas, agua, electricidad) permitiendo reducir costes [8][9].

2.1.6 Entorno AmILab

Figura 2-5: Logo AmILab

Entorno sobre el cual se centra este proyecto, AmILab (*Ambient Intelligence Laboratory*) consta de un laboratorio que sirve de plataforma de pruebas para probar desarrollos en tecnologías de entornos inteligentes. Se centra en la implementación de un sistema, denominado *blackboard* o pizarra, de agentes modulares y configurables basados en reglas que permiten explicar comportamientos y adaptarse a las necesidades cambiantes de los usuarios.

Dichas reglas, como se verá más adelante en mayor detalle, constan de eventos, condiciones y acciones que permiten expresar el comportamiento deseado del entorno así como permitir expresar el contexto de bajo nivel por medio de un lenguaje de alto nivel. El objetivo final del sistema es poner el control del entorno en las manos del usuario, permitiendo crear diseños complejos de manera sencilla y sin asistencia de expertos [10][11][12].

2.1.7 Conclusiones

Como se puede ver, la motivación común de los proyectos mencionados es el deseo de simplificar, en medida de lo posible, un entorno que por definición requiere de habilidades de múltiples disciplinas para su desarrollo y diseño.

Por un lado se tienen las dificultades planteadas por la existencia de dispositivos muy distintos que requieren de un sistema capaz de operar con ellos de manera indiferente y adaptable, siendo el enfoque de proyectos como **AMIGO** o **GTSH**, donde se busca facilitar la tarea de los diseñadores y programadores del entorno, en una capa que de otra forma requeriría diversas especializaciones.

Por otro lado, se busca llegar al usuario final, simplificando las interfaces sobre las cuales el usuario opera sobre el entorno, como es el caso de los mandos desarrollados en el proyecto sobre **Matilda's Smart House**, o el enfoque orientado a las personas investigado por la iniciativa AHRI en el **Georgia Aware Home**.

Siguiendo motivaciones similares, el proyecto **SMARTLAB** y los proyectos desarrollados en el entorno propio **AmILab**, buscan conseguir solucionar ambas dificultades planteadas, así como en el caso de la segunda, llevarlo a un paso adicional, permitiendo que el diseño del entorno pase de manos de diseñadores y programadores al usuario de la manera más sencilla posible.

2.2 Herramientas para la personalización de entornos

2.2.1 DaFRule

Proyecto desarrollado en la Universidad Politécnica de Valencia que tiene por objetivo la creación de un modelo que permita la edición de reglas de comportamiento de manera genérica. Dicho modelo permite la creación de reglas de forma independiente al dominio de aplicación, en el caso de este proyecto, la personalización de entornos de Inteligencia Ambiental y la especificación de comportamientos reactivos en entidades de juego.

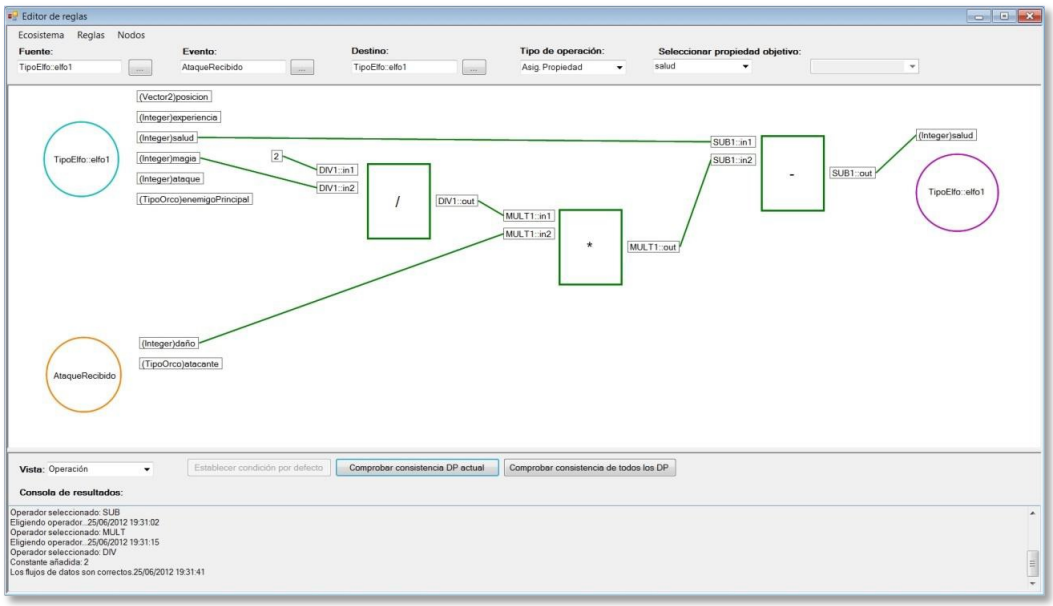

Figura 2-6: Editor de reglas DaFRule

Para la prueba del modelo se ha creado una herramienta prototipo que tiene por función el editado de reglas de un forma intuitiva, obviando la necesidad de la escritura de código textual por parte del usuario. El mecanismo de definición de reglas de manera visual resulta sencillo de comprender para usuarios sin conocimientos previos de programación, pero sin perder expresividad por ello, ya que la aproximación mediante flujos de datos hace posible combinar operadores entre sí, mientras que el modelado de los procesos de datos permite ampliar la cantidad de operadores disponibles en función del ámbito de aplicación [13].

2.2.2 Nexel

Diseñado por el *Institute of Parallel and Distributes System* de la universidad Stuttgart, **Nexel** es un lenguaje de programación gráfico que busca simplificar el desarrollo de pequeñas aplicaciones en el ámbito de Inteligencia Ambiental. El editor desarrollado permite al usuario especificar comportamientos que se deben realizar como resultado de eventos producidos por dispositivos del entorno.

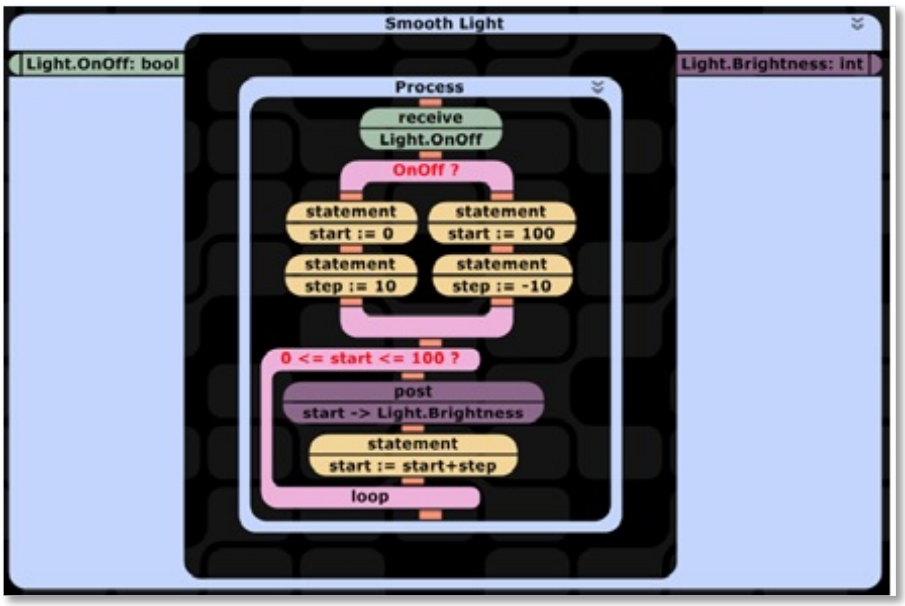

Figura 2-7: Editor de reglas Nexel

Dichos comportamientos se determinan visualmente mediante una secuencia de instrucciones asociadas al evento mediante su arrastre a las posiciones deseadas, con posibilidad de añadir complejidad por medio de bifurcaciones en el flujo y estructuras de repetición [14].

2.2.3 a CAPpella

Este proyecto se centra en el usuario final, buscando permitir el definir comportamientos basados en una situación específica definida por éste, sin llegar a escribir ningún código. Se busca que el usuario demuestre el comportamiento al sistema, anotando las partes relevantes, y que el sistema "aprenda" de dicha forma, por medio de varios ejemplos, la manera de actuar frente al contexto mostrado.

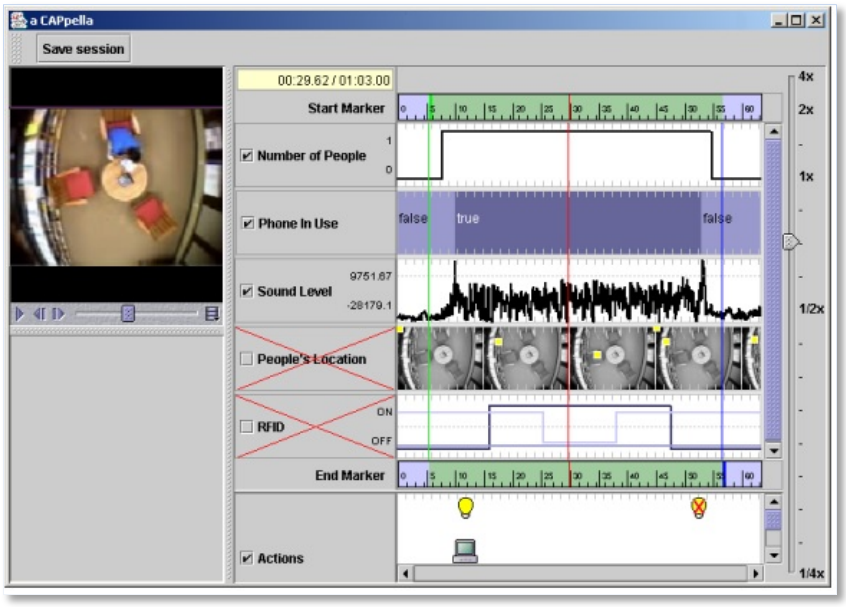

Figura 2-8: Herramienta a CAPpella

Mediante esta programación por demostración, la interfaz del usuario se reduce a una herramienta que permite el grabado de sucesos en el entorno referidos a un contexto determinado (señalando las entradas relevantes) que luego es analizado para determinar las acciones pertinentes que se desean automatizar [15].

2.2.4 Conclusiones

El proyecto realizado busca crear una herramienta para permitir al usuario final diseñar comportamientos basados en un sistema de reglas ECA (evento-condición-acción) que actúan sobre un entorno inteligente, demostrado como una de las formas más intuitivas para el usuario final de programar [16]. Desde esta perspectiva, **DaFRule** se basa en la creación de un sistema de reglas siguiendo la misma estructura, elaborando una herramienta que permite su diseño por medio de un diagrama de flujos. Esta solución permite una visualización intuitiva al usuario, pero debido al enfoque genérico del proyecto, es decir, busca ser independiente de la aplicación final de las reglas definidas, no permite el grado de integración visual con el entorno que daría un mayor nivel de familiaridad al usuario.

Por otro lado, las herramientas definidas por los proyectos **Nexel** y **a CAPpella** están diseñadas con la aplicación en entornos inteligentes en mente, si bien la primera, **Nexel**, busca principalmente facilitar el desarrollo de aplicaciones sobre la plataforma, enfocándose por tanto más en diseñadores y programadores que en el usuario final, y requiriendo a su vez un conocimiento mayor en lenguajes de programación para su uso. El proyecto **a CAPpella** resulta interesante debido a su acercamiento al problema de una forma diferente, desechando un sistema basado en reglas desde la perspectiva del usuario, permitiendo una manera intuitiva y desligada de la programación de definir comportamientos, aún perdiendo parte de la eficiencia y precisión aportada por dichos métodos.

3 Programando comportamientos con AmIRules

Como se ha mencionado, el entorno sobre el cual trabaja este proyecto se basa en una arquitectura denominada pizarra o *blackboard*. Dicha arquitectura consta de una base de datos distribuida en entidades, representaciones de los elementos que componen el entorno, cada una con sus propiedades, relaciones con otras entidades y capacidades sobre las que actuar. Cada cambio en el entorno actualiza dichos valores, y viceversa, un cambio sobre la capacidades conlleva un cambio en el entorno.

Para permitir al usuario personalizar el entorno de una manera eficiente, la pizarra consta de intérpretes de reglas, independientes entre sí, denominados agentes. Dichas reglas siguen un modelo de evento-condición-acción (ECA-rules), siendo capaces de describir comportamientos complejos de forma sencilla. A cada agente, determinado por el usuario, se le asigna un conjunto de reglas sobre el cuál operar y es responsable de monitorizar los eventos que activan dichas reglas, las condiciones sobre las que actuar, y de realizar las acciones pertinentes a cada ejecución.

A continuación se procederá a explicar en detalle la lógica y sintaxis de las reglas AmIRules, así como el funcionamiento interno de los agentes [17].

3.1 Descripción de reglas

Una regla genérica toma la forma que se indica en la Figura 3-1.

```
evento1 || evento2 || ...
  ::
condición1 && condición2 && ...
 =>
acción1 && acción2 && ...
;
```

Figura 3-1: Regla genérica AmIRules

En donde "||" y "*&&*" indican disyunción y conjunción respectivamente, "*::*" delimita los eventos de las condiciones y "=>" indica el paso a las acciones, con "*;*" finalizando la regla.

Como se puede ver, por tanto, las reglas se dividen en tres partes fundamentales, eventos o *triggers*, condiciones y acciones, que se pasan a detallar en las siguientes secciones.

3.1.1 Evento

Variable de contexto supervisada por el agente que indica cuándo se debe activar la regla al producirse un cambio en la misma. Puede ser la propiedad de una entidad del entorno o la creación (o eliminación) de una relación entre distintas entidades.

En el caso de múltiples eventos, se permite la disyunción de eventos simples en una misma regla. La creación de eventos complejos es posible a través de la concatenación de múltiples reglas mediante el uso del temporizador, descrito más adelante.

3.1.2 Condición

Conjunto formado entre una variable de contexto (la propiedad de una entidad o relación de varias entidades), un comparador y un valor asignado (u otra variable), que debe satisfacerse para producirse la consecuente ejecución de las acciones. Funcionalmente, las condiciones codifican el contexto de las acciones. Se permite la conjunción de diversas condiciones, la disyuntiva siendo descrita por medio de reglas separadas. También es posible omitir la existencia de condiciones, dando lugar a reglas de evento-acción (EA-rules).

Las condiciones soportan el siguiente grupo de comparadores:

- = : Retorna verdadero si ambos elementos a los lados del comparador son iguales. Al comparar una relación a un valor, retorna verdadero si cualquier relación del subconjunto coincide con el valor especificado. En el caso de comparar dos relaciones entre sí, retorna verdadero si la relación al lado izquierdo es un subconjunto de la relación al lado derecho.
- *!*= : Realiza la operación contraria al comparador anterior.
- > : Sólo válido para enteros, retorna verdadero si el valor izquierdo es mayor que el valor derecho.
- < : Sólo válido para enteros, retorna verdadero si el valor derecho es mayor que el valor izquierdo.
- >= : Sólo válido para enteros, retorna verdadero si el valor izquierdo es mayor o igual que el valor derecho.
- <= : Sólo válido para enteros, retorna verdadero si el valor derecho es mayor o igual que el valor izquierdo.

3.1.3 Acción

De forma análoga a las condiciones, conjunto formado por una variable de contexto y un valor determinado, a asignarse en el caso de que se den las condiciones especificadas. Toda acción se representa como un cambio en la base de datos de entidades de la pizarra, con las propiedades adecuadas indicando a ésta como debe proceder.

Teniendo en cuenta las acciones como actualizaciones de estado, se permiten las siguientes operaciones:

- *CE* : Crea una nueva entidad en la pizarra. Sigue la estructura de <tipo de entidad> CE <nombre de la entidad>, donde el nombre puede emplear el carácter "#" para generar un identificador único.
- *AP* : Añade una nueva propiedad (de tipo opcional) a una entidad ya existente. Si una entidad fue creada en misma regla con un identificador generado, "#" corresponderá a dicho identificador.
- -> : Añade una relación a una entidad.

- -< : Elimina una relación existente.
- *:*= : Asigna un valor a una propiedad.
- *=!* : Asigna el valor contrario (sólo válido para booleanos).
- =- : Substrae de una propiedad (sólo válido para enteros) un valor directo o el valor de otra propiedad.
- += : Substrae de una propiedad (sólo válido para enteros) un valor directo o el valor de otra propiedad.

3.1.4 Comodines o *Wildcards*

Los eventos, las condiciones y las acciones, según se ha visto, operan sobre propiedades o relaciones de entidades determinadas. Para extender la funcionalidad de las reglas a expresiones genéricas se introducen los símbolos de "*", expresando toda entidad de un mismo tipo, y "*$*", expresando el grupo de entidades derivado de cumplirse el evento o condición sobre la entidad genérica especificada, denominados comodines o *wildcards*.

De esta forma, para indicar que un evento se puede producir por cualquier entidad de un tipo particular, se sustituye el nombre de entidad por "*", empleándose "*$*" seguido de un índice (relativo al orden de aparición del "*" en la regla, empezando en 0) para indicar la entidad específica que activó el evento en expresiones sucesivas. Por otro lado, una condición sobre el valor de la propiedad de una entidad genérica dará verdadero siempre y cuando haya al menos una entidad del tipo adecuado que la cumpla, con "*$*" expresando el conjunto filtrado final.

En la Figura 3-2 se muestra un ejemplo de la aplicación de *wildcards* en la creación de una regla que actúa sobre grupos de entidades. En este caso, se busca encender cualquier luz apagada ubicada en una localización a la que acaba de entrar una persona cualquiera.

```
Person:*@amilab!locatedAt
 ::
Light:*@amilab!locatedAt = Person:$0@amilab!locatedAt
 &&
Light:$1@amilab!isSwitchable/status.binaryValue = 0
 =>
Light:$1@amilab!isSwitchable/status.binaryValue := 1
 ;
```

Figura 3-2: Ejemplo del uso de *wildcards*

3.1.5 Tiempo

Como se ha mencionado anteriormente, para la creación de reglas complejas se hace uso del temporizador. Dicho temporizador permite la expresión de acciones basadas en el tiempo formando una acción en una regla de la forma que se muestran en la Figura 3-3.

```
TIMER ending_time concurrence
{
        on_finished_rules
}
{
        on_running_rules
}
{
        on_load_rules
}
```

Figura 3-3: Estructura de una acción basada en el tiempo

En donde:

- *ending_time* se refiere al intervalo de tiempo, ya sea especificando el tiempo de parada determinado, el tiempo activo en relación al tiempo de inicio o infinito, en cuyo caso debe finalizarse por la llamada apropiada, descrita a continuación.

- *concurrence* determina el número de temporizadores que pueden ejecutarse simultáneamente.
- *on_finished_rules* son el conjunto de reglas CA (condición-acción) a ejecutarse al finalizar el temporizador.
- *on_running_rules* conforman el conjunto de reglas ECA activas durante el periodo activo del temporizador, que pueden a su vez actuar sobre el estado del temporizador.
- *on_load_rules* son el conjunto opcional de reglas CA a ejecutar cuando el temporizador es iniciado.

Para permitir a las reglas del conjunto *on_running_rules* actuar sobre el temporizador se definen las siguientes acciones permitidas:

- *TIMER.pause* : El temporizador permanece activo pero la cuenta hacia el tiempo de acabado se detiene.
- *TIMER.play* : Continúa la cuenta atrás desde donde se detuvo.
- *TIMER.stop* : Se finaliza la cuenta atrás, ejecutándose las reglas del conjunto *on_finished_rules*.
- *TIMER.kill* : Se fuerza el temporizador a finalizar, evitando, por tanto, que se ejecuten las reglas *on_finished_rules*.
- *TIMER.start* : Evento generado al iniciarse el temporizador.
- *TIMER.replay* : La cuenta se reinicia, generándose el evento *TIMER.start*.
- *TIMER.reset* : Se fuerza el temporizador a reiniciarse, volviéndose a ejecutar las reglas del conjunto *on_load_rules*.

En la Figura 3-4 se puede ver un ejemplo del uso del temporizador para crear una regla con el objetivo de apagar la luz al pasar un minuto desde que *javierge* abandona la habitación (se omiten las reglas opcionales *on_load* y se cancela el temporizador si vuelve a entrar en la habitación antes de que transcurra el tiempo).

```
!Person:javierge@amilab!locatedAt/Room:lab_B403_Office
::
Light:lamp2@amilab!isSwitchable/status.binaryValue=1
=>
TIMER 1m 1
{
        Light:lamp2@amilab!isSwitchable/status.binaryValue:=0;
}
        Person:javierge@amilab!locatedAt/Room:lab_B403_Office
        :: =>
        TIMER.kill
        ;
}
;
```

Figura 3-4: Ejemplo del uso del temporizador

3.2 Estructura de agentes

Los agentes conforman entidades independientes de la pizarra cuya función es supervisar y ejecutar conjuntos de reglas. Para realizar dicha tarea los agentes se dividen en dos partes: por un lado la clase principal **Agente** y, por otro, el *thread* iniciado en la anterior o *callback*, **RCallback_Agente**.

3.2.1 Agente

La clase principal consta de los procedimientos que se deben seguir en el arrancado del agente:

- Lectura del fichero de configuración del agente, donde debe especificarse el nombre de usuario y contraseña a utilizarse para acceder a la pizarra así como, opcionalmente, ya que se pueden pasar por argumentos, el nombre del agente, el puerto a usar y el fichero de reglas que se debe cargar.
- Creación de la entidad que reflejará el estado actual del agente en la pizarra.
- Arranque del *thread* que ejecutará la clase *callback* en caso de haberse creado la entidad de la pizarra sin problemas, en caso contrario se vuelve a intentar el paso anterior (con un número limitado de intentos).

- *ShutdownHook* encargado de cerrar correctamente el agente, eliminando las subscripciones a la pizarra pertinentes.
- Consola de comandos, que permite a un usuario directo una forma de control sobre el agente, además de servir como un canal de control para otros procesos externos (p.ej enviar el comando de cerrar desde la aplicación de control de agentes).

3.2.2 Callback

La clase *callback* es la encargada de supervisar la activación de los eventos descritos en las reglas asociadas al agente. Para llevar a cabo este proceso se comienza con la lectura del fichero de reglas y la consecuente subscripción a los eventos de la pizarra determinados en ellas, de esta forma, la pizarra hará saber al *callback* cuando se produce un cambio a la entidades subscritas y se podrá proceder a la comprobación de las condiciones, para luego, en caso positivo, comunicar a la pizarra los cambios que debe realizar para ejecutar las acciones especificadas.

3.3 Conclusiones

Como se ha visto, las reglas AmIRules permiten programar comportamientos complejos empleando una estructura sencilla, lo cual permite generar una interfaz intuitiva que describa dichos comportamientos, y ponerlos en funcionamiento a través de agentes dedicados. A su vez, dichos agentes pueden verse conceptualmente como simples agrupaciones de reglas, lo cual conlleva a simplificar considerablemente el sistema a nivel de usuario.

A continuación se pasará a describir la aplicación encargada de llevar a cabo dicha tarea.

4 BBb: Arquitectura software de la herramienta

4.1 Descripción de alto nivel

La herramienta diseñada busca ofrecer una interfaz simple al usuario para la creación y posterior control de agentes sobre el entorno de la pizarra, ofreciendo una solución para la personalización de entornos inteligentes. Con esta finalidad se ha optado por el desarrollo de una aplicación web en donde el servidor ofrece un canal a la información contenida en la pizarra, así como control sobre el conjunto de agentes creados y organizados bajo su supervisión, mientras que el cliente permite exponer dicha información de manera intuitiva.

Por otro lado, se dispone de un módulo adicional dedicado a la facilitación del proceso de creación de reglas basadas en eventos de localización. Dicho módulo emplea una estructura cliente-servidor similar a la herramienta principal y sirve de ejemplo para otros posibles módulos de diseño de reglas con posibilidad de añadirse a la herramienta principal en un futuro.

De esta forma, las herramientas diseñadas se relacionan con la arquitectura del entorno como constructores de reglas y agentes, agentes que manipulan el estado de las entidades (de las cuales también son parte) que están contenidas en la pizarra (Figura 4-1).

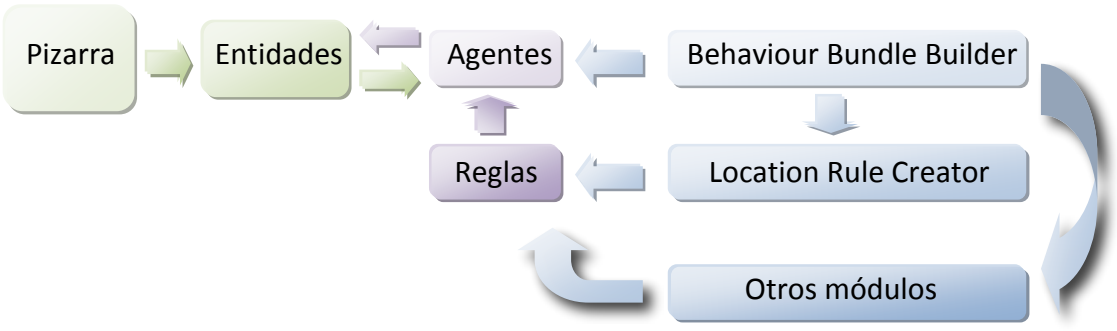

Figura 4-1: Arquitectura del entorno respecto a las herramientas

4.2 Descripción de operaciones por parte del usuario

Desde el punto de vista de funcionalidad por parte del usuario, la herramienta principal puede dividirse en tres partes: administración de agentes, administración de reglas y edición de reglas.

En cuanto a la administración de agentes, se permiten una serie de operaciones sobre los agentes no activos en la pizarra:

- Crear nuevos agentes, dándoles el nombre deseado o asignándole un valor por defecto. Dicha acción crea un nuevo archivo que contendrá las reglas del agente generado cuando se realice su activación.
- Renombrar agentes existentes, renombrando el fichero del cuál coge el nombre el agente generado.
- Eliminar agentes existentes, borrando el fichero de reglas asociado.
- Activar el agente, generando un nuevo proceso con el agente en la pizarra a partir del fichero de reglas que lo representa.
- Editar el agente, permitiendo las operaciones descritas más adelante.

Los agentes activos, por su parte, solo pueden ser desactivados ya que no se permite la edición de su fichero de reglas una vez se encuentran generados en la pizarra.

La administración de reglas, dado lugar por la edición de un fichero de reglas (que representa un agente) permiten las siguientes operaciones:

- Crear nuevas reglas, de nuevo asignando o bien un nombre determinado o un valor por defecto. Esta acción crea nuevas entradas en el fichero de reglas abierto una vez se guarden los cambios realizados.
- Renombrar reglas, cambiando la entrada correspondiente del fichero una vez se guarden los cambios realizados.
- Eliminar reglas, de la misma forma, modificando la entrada del fichero que se desea eliminar una vez guardados los cambios.

- Mover reglas a otros agentes, a diferencia de las anteriores operaciones, guardando los cambios automáticamente (después de confirmación) y creando/eliminando las entradas correspondientes en los ficheros de reglas o agentes determinados.
- Guardar cambios, realizando las operaciones escogidas sobre el fichero abierto.
- *Reload*, deshaciendo los cambios realizados cargando el fichero de nuevo sin modificar.
- Cerrar el editado, cancelando todos los cambios realizados que no se hayan guardado.
- Y, por último, editar la regla seleccionada, también a la espera del guardado.

La operación del editado de reglas permite el acceso a los módulos de creación de reglas que se hayan implementado o la edición directa de la regla seleccionada mediante su división en partes fundamentales (evento-condiciones-acciones).

4.3 Descripción estructural

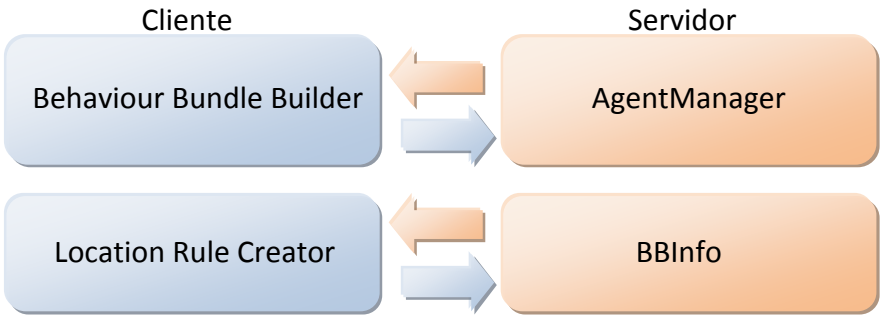

Figura 4-2: Estructura cliente-servidor

Como se puede ver en la Figura 4-2 y como se ha descrito con anterioridad, se tiene un modelo cliente-servidor, en donde dos clases principales, **AgentManager** y **BBInfo**, descritas a continuación, actúan de servidores o *servlets* para la herramienta de administración de agentes y para el creador de reglas de localización, respectivamente.

4.3.1 AgentManager

La clase **AgentManager** tiene por función almacenar los procesos pertenecientes a cada agente en ejecución por la herramienta, el fichero de reglas sobre el cual se está editando, así como los puertos disponibles y en uso, además de operar directamente sobre los archivos de reglas que determinan los distintos agentes. Da acceso a los siguientes métodos [ANEXO]:

- **getBundles.** Devuelve los nombres de los agentes que supervisa la herramienta, es decir, los nombres de los ficheros de reglas.
- **getActiveBundles.** Devuelve los nombres de los agentes supervisados que están actualmente activos en la pizarra.
- **newAgent.** Crea un nuevo fichero de reglas correspondiente a un nuevo agente. Se hacen comprobaciones para nombres ya existentes.
- **renameAgent.** Cambia el nombre de un agente existente y, por tanto, el nombre del fichero de reglas asociado. Se hacen comprobaciones para nombres ya existentes.
- **removeAgent.** Elimina el fichero de reglas que representa a un agente.
- **startAgent.** Crea un nuevo proceso que arranca el agente con el fichero de reglas especificado, asignándose un puerto de la lista de puertos accesibles.
- **stopAgent.** Envía el comando de cerrar a la consola del proceso del agente determinado.
- **checkAgent.** Comprueba si el agente/s dado está ya activo, está siendo editado y si hay puertos disponibles.
- **loadRulesFile.** Carga un fichero de reglas al mapa de reglas del servidor para ser editado.
- **saveRulesFile.** Guarda las reglas editadas en el fichero de reglas correspondiente.
- **closeRulesFile.** Limpia el mapa de reglas del servidor.
- **getRulesFromMap.** Devuelve la lista de nombres de las reglas contenidas en el mapa de reglas del servidor.
- **getRule.** Devuelve el contenido de la regla, especificada por nombre, del mapa de reglas.
- **setRule.** Almacena la regla editada en el mapa de reglas.

- **newRule.** Crea una nueva regla en el mapa de reglas. Se hacen comprobaciones para nombres ya existentes y se emplea una nomenclatura por defecto en caso positivo.
- **moveRule.** Mueve una regla del mapa de reglas a otro fichero de reglas ya existente. Se guardan los cambios realizados en ambos ficheros. Se hacen comprobaciones para nombres ya existentes y se emplea una nomenclatura por defecto en caso positivo.
- **renameRule.** Cambia el nombre de una regla en el mapa de reglas.
- **removeRule.** Elimina una regla existente del mapa de reglas.
- **checkRuleName.** Comprueba si existe una regla del nombre dado en el mapa de reglas.

4.3.2 BBInfo

La clase **BBInfo** ofrece una variedad de métodos que permiten acceder a información procedente de la pizarra necesaria para la creación de reglas coherentes con el entorno [ANEXO].

- **getEntities.** Devuelve las entidades pertenecientes al dominio de trabajo de la pizarra.
- **getRangeEntities.** Devuelve las entidades con posibilidad de actuar de sujeto en el evento de localización, en este caso, de tipo *person*.
- **getFirstEntities.** Devuelve las entidades de localización iniciales desde las cuales se podrá avanzar a otras localizaciones por relaciones.
- **getLocationComposition.** Devuelve las localizaciones de las que está compuesta una localización inicial.
- **getIsComposedOf.** Comprueba si una localización está compuesta de otra dada.
- **exitLocation.** Devuelve la localización que contiene la actual.
- **getAreaEntities.** Devuelve las entidades que se encuentran en una localización especificada.
- **getEntityProperties.** Devuelve las propiedades de la entidad determinada por el *path* dado.

- **getEntityCapabilities.** Devuelve las capacidades de la entidad determinada por el *path* dado.
- **getEntityCapabilityValues.** Devuelve los posibles valores de la capacidad específicada.
- **getEntityRelations.** Devuelve las relaciones de la entidad determinada por el *path* dado.

4.4 Descripción de comportamiento

Para acceder a la información y operaciones ofrecidas por las funciones de los *servlets* descritos en el apartado anterior, las páginas clientes *JSP*, tanto en la herramienta de control de agentes como en el creador de reglas de localización, emplean llamadas *Ajax* desde funciones *Javascript* o eventos, procesando la respuesta en otras funciones *Javascript* que afectarán al contenido de la página o repetirán el ciclo realizando la llamada *Ajax* necesaria. En la Figura 4-3 se representa un diagrama del proceso seguido.

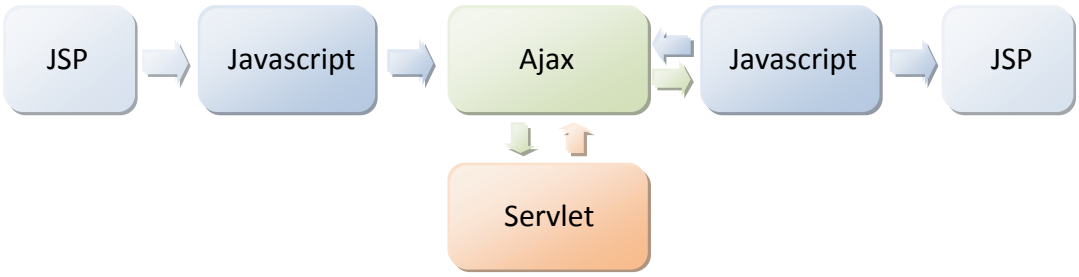

Figura 4-3: Proceso de acceso a la información del servidor

Como ejemplo complejo de dicho comportamiento se describirá a continuación el proceso seguido para el arranque de un agente en la pizarra:

- Se produce el evento *onclick* del botón **a_start**, asociado al arranque de uno o varios agentes seleccionados, que ejecuta la función *javascript* **splitBundlesThen** con argumento '*checkAgent*'.

- La función **splitBundlesThen** divide los agentes seleccionados y los anexa en un *string* con delimitadores determinados para luego realizar una llamada a la función *Ajax* con el *string* y el nombre de la función determinada en el argumento, en este caso '*checkAgent*'.
- El servlet **AgentManager** recibe el *request* de la función *Ajax* con los argumentos dados en la llamada, e interpreta que debe llamar a la función **checkAgent** con el *string* dado como parámetro.
- **checkAgent**, como se ha visto, comprueba si los agentes de la lista en el *string* se encuentran ya activos, están siendo editados y si hay puertos disponibles para arrancar más agentes. Devuelve un *string* con las respuestas booleanas concatenadas.
- El *string* devuelto por el *servlet* es recogido por la función *Ajax* que en este caso llama a otra función *Javascript* **startAgent**, con la respuesta del *servlet* como argumento.
- La función **startAgent** comprueba el *string* recibido y procede en el caso de que los agentes especificados no están siendo editados y existen puertos disponibles para iniciarlos todos, volviendo a dividir la lista de agentes en un *string* y llamando a la función *Ajax* con argumento '*startAgent'*. En caso contrario realiza el *alert* con la descripción del error adecuado.
- De nuevo, el *servlet* recibe el *request* con el *string* de argumento y con nombre de la función a llamar, en este caso '*statAgent'*.
- **startAgent**, además de arrancar los agentes en procesos individuales y añadirlos a la lista pertinente, realiza previamente una comprobación de los agentes ya activos y simplemente los ignora en el *string* recibido.
- La función *Ajax* no recibe en este caso información del *servlet* ya que es una función de operación, pero procede a llamarse a sí misma con argumento '*loadBundles*'.
- Desde el *servlet*, se recibe **loadBundles** como función, lo cual se interpreta como un *string* concatenado de las respuestas de las funciones **getBundles** y **getActiveBundles**.
- La función *Ajax* recibe dicho *string* y lo pasa a la función *Javascript* **loadBundles** que será la encargada de actualizar las listas de la página con los agentes activos y no activos.

4.5 Conclusiones

La estructura que se ha empleado permite llevar a cabo ampliaciones futuras de una manera sencilla, ya sea, como se ha comentado anteriormente, añadiendo nuevos módulos para la creación de reglas, o nuevas funcionalidades a las herramientas ya existentes.
Por otro lado, el diseño de la interfaz de cada módulo queda ligado al cliente, generado por la página *JSP*, con su correspondiente fichero de estilo *CSS* y funciones *Javascript*.

A continuación se describirán dichas interfaces, tanto para la herramienta de control de agentes, como para el módulo de creación de reglas por localización.

5 BBb: Interfaz gráfica de usuario

5.1 Behaviour Bundle builder

La herramienta de control de agentes, **Behaviour Bundle builder** o constructor de agrupaciones de comportamientos, se divide principalmente en dos partes en cuanto a funcionalidad e interfaz: por un lado, se encuentra el propio control de los *bundles* o agentes y, por otro, el editado de las reglas de comportamiento contenidas en dichos *bundles*.

En el siguiente apartado se describe la interfaz de usuario asociada a la primera funcionalidad mencionada:

5.1.1 Control de agentes o *bundles*

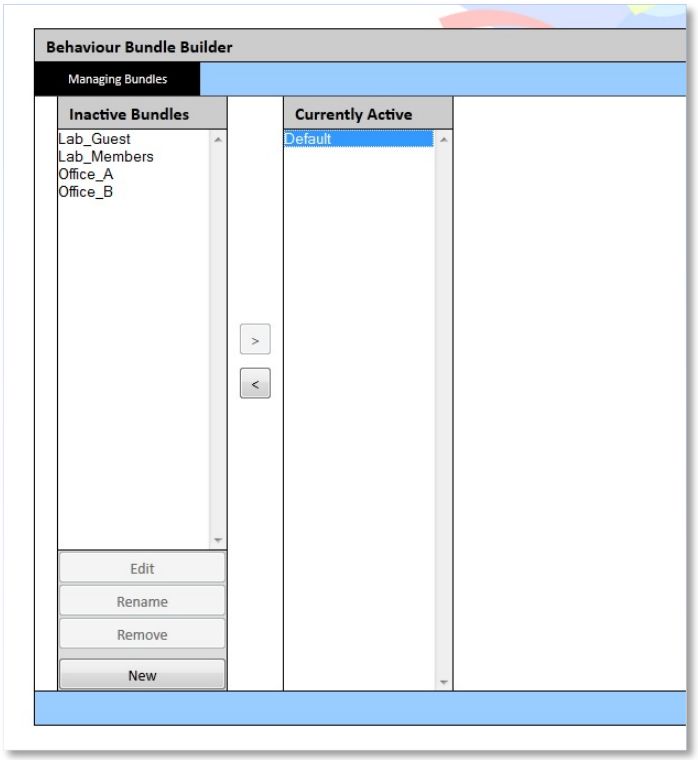

Figura 5-1: Interfaz BBb, "Managing Bundles"

Como se puede ver en la Figura 5-1, la vista se divide principalmente en dos componentes, *inactive bundles*, que contiene la lista de los *bundles* de reglas existentes en el fichero que se pueden activar o editar, y *currently active*, que contiene la lista de aquellos *bundles* asociados a la herramienta que ya se encuentran activos en el sistema de la pizarra o *blackboard*.

Los controles relacionados con *inactive bundles* son aquellos pertenecientes al panel de control inferior (Figura 5-2), que permiten las siguientes acciones: el editado del *bundle* seleccionado, cambiar el nombre siempre y cuando no esté siendo editado, eliminar varios *bundles* con la misma condición anterior, y la creación de nuevos *bundles*, permitiendo escoger el nombre a usar mediante un cuadro de texto que toma el lugar del botón asociado (en caso de un nombre en blanco se utilizará un nombre genérico con numeración basada en los ya existentes).

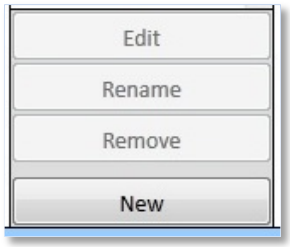

Figura 5-2: Interfaz BBb, control de *inactive bundles*

Además de los controles relacionados al editado, se tiene el control de activación (Figura 5-3), representado por la flecha entre ambos selectores y que permite el activado de uno o más *bundles* simultáneos en la pizarra, si se cumple que no están siendo ya editados y hay suficientes puertos (determinado por el archivo de configuración) para soportarlos a todos.

Figura 5-3: Interfaz BBb, control de activación de *bundles*

En contraste, los *bundles* de *currently active* solo permiten su desactivación por medio de la flecha correspondiente, también de manera múltiple y simultánea, dado que no se permite el editado de un *bundle* ya activo en la pizarra.

5.1.2 Editado de agentes o *bundles*

Una vez se ha iniciado el editado de un *bundle*, la interfaz pertinente al control de agentes se deshabilita y se pasa a mostrar una lista de las reglas contenidas en el agente seleccionado como se puede apreciar en la Figura 5-4.

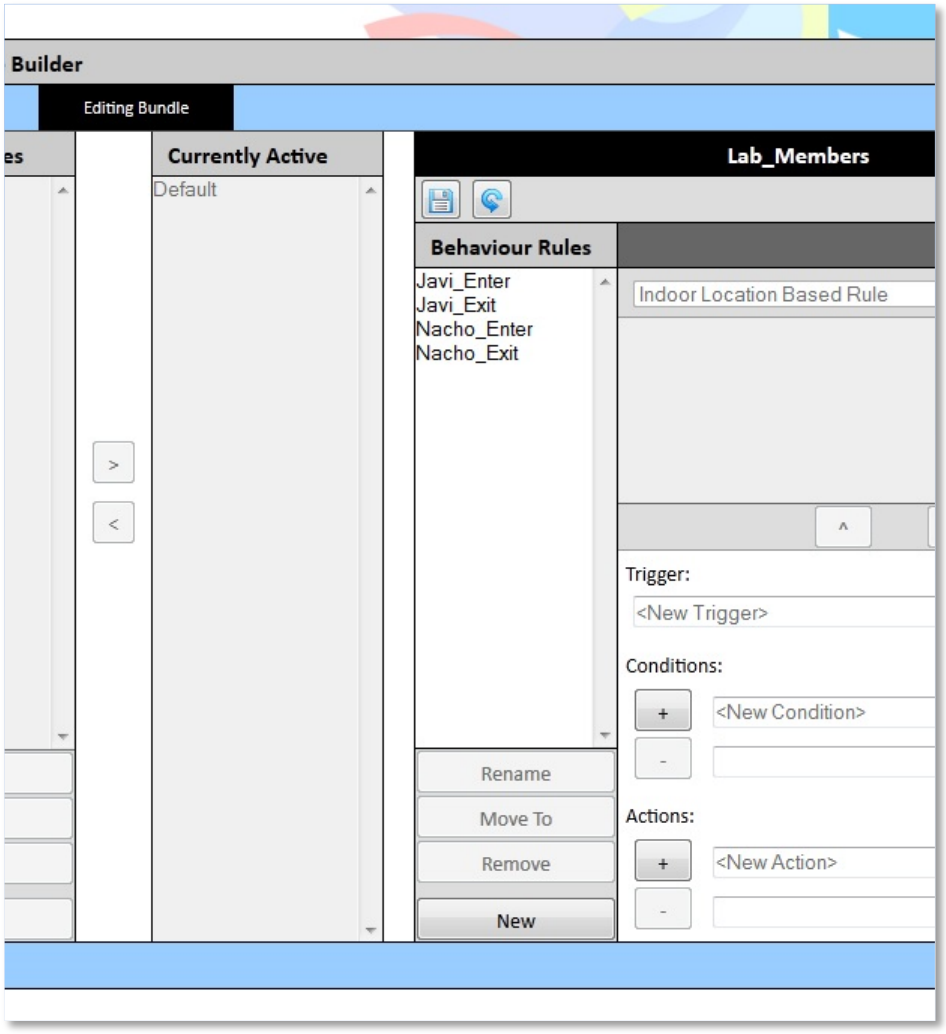

Figura 5-4: Interfaz BBb, "Editing Bundle"

De forma análoga al selector de *inactive bundles* del apartado anterior, las acciones a realizar con las reglas vienen dadas en el panel de control inferior (Figura 5-5), con la excepción del editado, que se accede de manera directa al seleccionar una regla.

Las funciones de renombrado, eliminado y creación de nuevas reglas funcionan de forma similar a las equivalentes en el control de *bundles*, si bien los resultados de dichas acciones no se guardan de forma automática en el fichero de reglas hasta que no se confirman los cambios, por medio del botón de guardado superior.

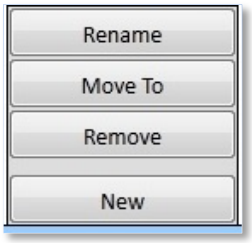

Figura 5-5: Interfaz BBb, control sobre reglas

Por otro lado, la función de mover reglas permite reorganizar reglas entre distintos *bundles*, manteniendo el nombre en el *bundle* destino de ser posible o, en caso contrario, empleando un nombre generado con un identificador numérico no en uso. A diferencia de las otras funciones, los cambios realizados por la función de mover son aplicados directamente a los ficheros de reglas correspondientes y, por tanto, requieren la confirmación del usuario.

5.1.3 Editado de reglas

Seleccionada una regla a editar, el contenido de la misma se muestra en el área de texto de forma directa, mostrado en la Figura 5-6. Para el editado del contenido se nos ofrecen dos opciones:

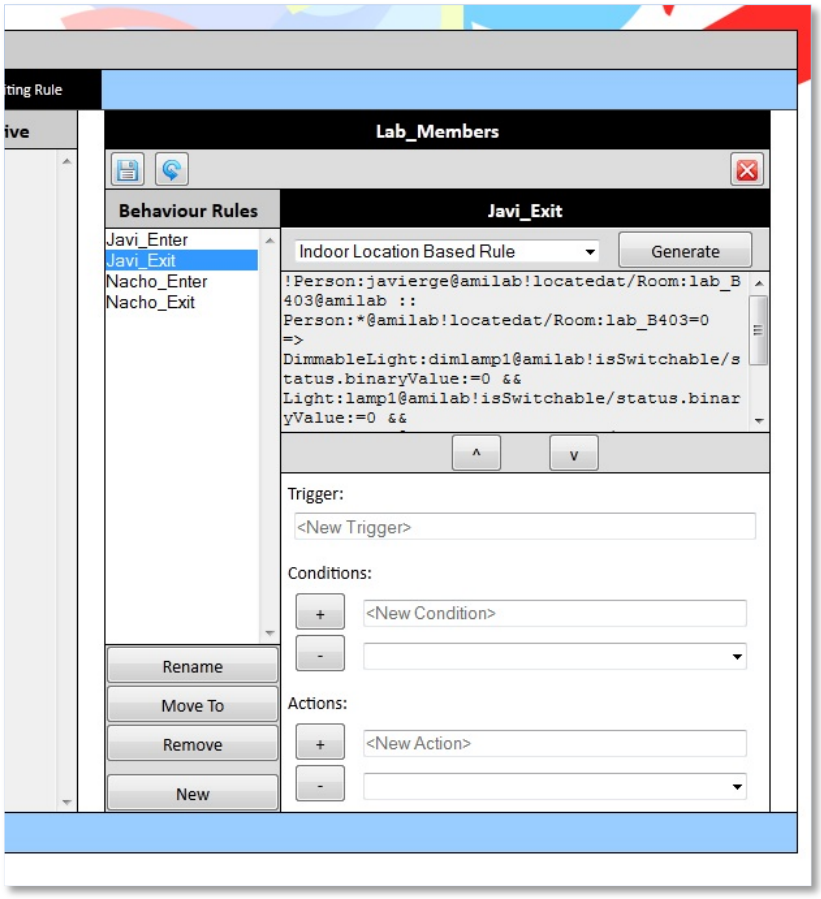

Figura 5-6: Interfaz BBb, "Editing Rule"

La primera, y preferida para los usuarios comunes, es el uso de una herramienta de generación de reglas, a seleccionar en el selector superior (Figura 5-7). Dicho selector permite acceder al **Indoor Location Based Rule Creator**, herramienta creada como ejemplo y descrita en detalle a continuación, y es ampliable de forma sencilla para acomodar a otras posibles herramientas en un futuro.

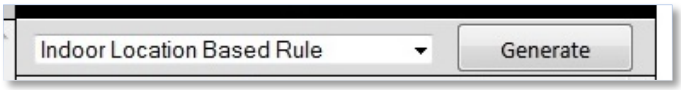

Figura 5-7: Interfaz BBb, selector de herramienta de edición de reglas

La segunda opción, para usuarios más avanzados, permite, mediante el uso de las flechas inferiores, modificar de manera más directa el contenido de la regla, separándola en sus partes básicas, evento, condiciones y acciones, y permitiendo el editado individual de cada una de las partes, como se puede ver en la Figura 5-8.

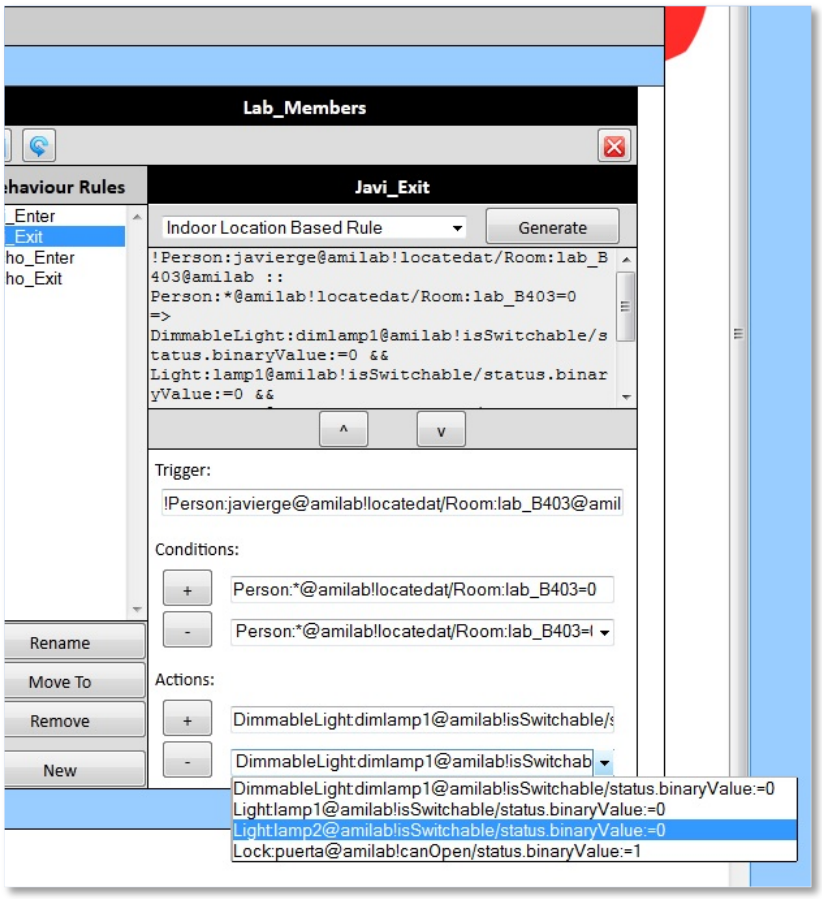

Figura 5-8: Interfaz BBb, editado manual de reglas

5.2 Indoor Location Based Rule Creator

El **Indoor Location Based Rule Creator**, o creador de reglas basadas en localización en interiores, es una herramienta que permite generar el código de una regla de forma sencilla con poco o ningún conocimiento sobre la sintaxis de la misma. Para llevar a cabo esta tarea, se guía al usuario a través de una serie de pasos, la especificación del evento que lanza la regla (en el caso de esta herramienta el contexto de localización permite reducir las posibilidades a dos posibles situaciones, entrada o salida de una entidad respecto de un determinado lugar), las condiciones a comprobar y las acciones a realizar.

5.2.1 Especificación del evento

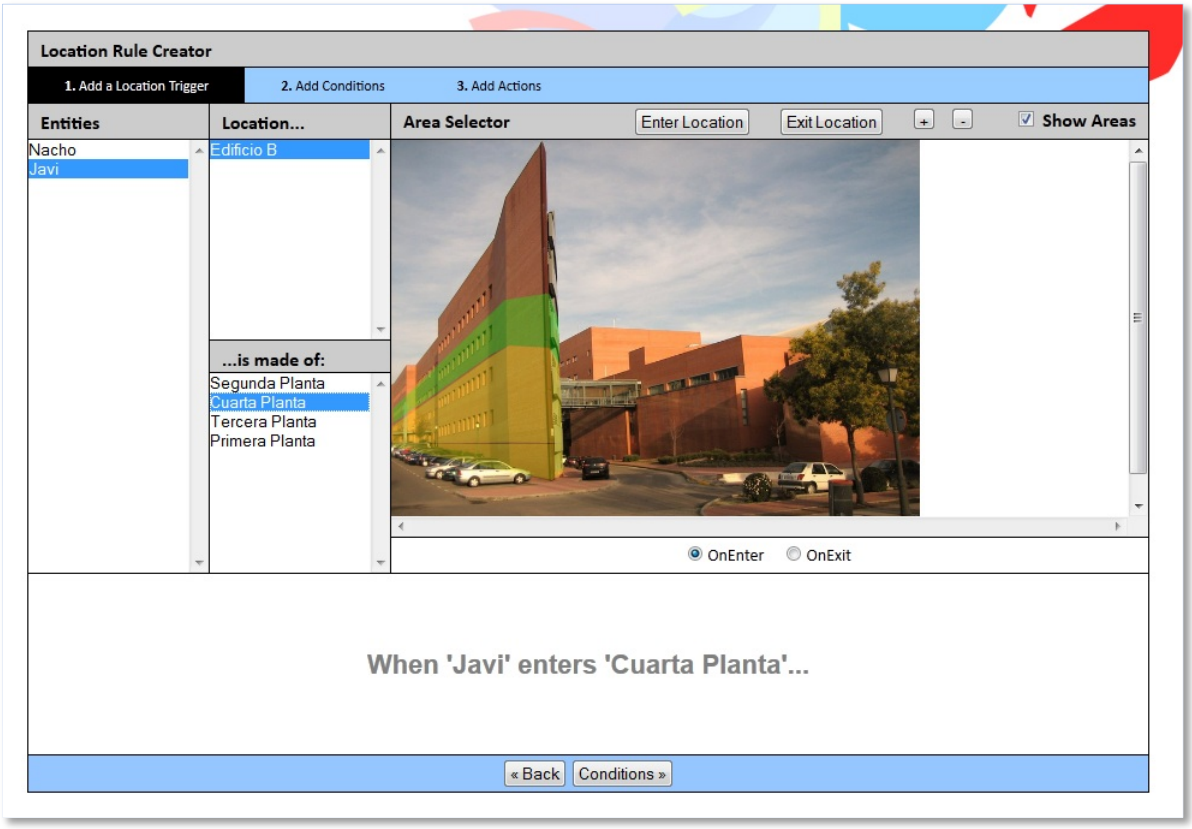

Figura 5-9: Interfaz ILB, configuración del evento o *trigger*

En la Figura 5-9 se muestra la configuración de un evento de una regla. Para generar dicho evento es necesario indicar, en primer lugar, la entidad referida mediante el selector apropiado *Entities* (Figura 5-10), que muestra todas las entidades del dominio de la pizarra filtradas por tipo *person* (como el nombre indica, entidades que representan a personas).

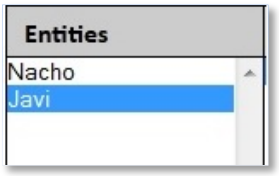

Figura 5-10: Interfaz ILB, selector de la entidad sujeto de la regla

En segundo lugar se especifica la localización (Figura 5-11), para ello se tiene el selector de *Location* además de la lista de interiores, *is made of*, que pertenecen al lugar seleccionado, empleándose los botones de *Enter Location* y *Exit Location* para navegar a través de las dependencias hasta seleccionar el destino.

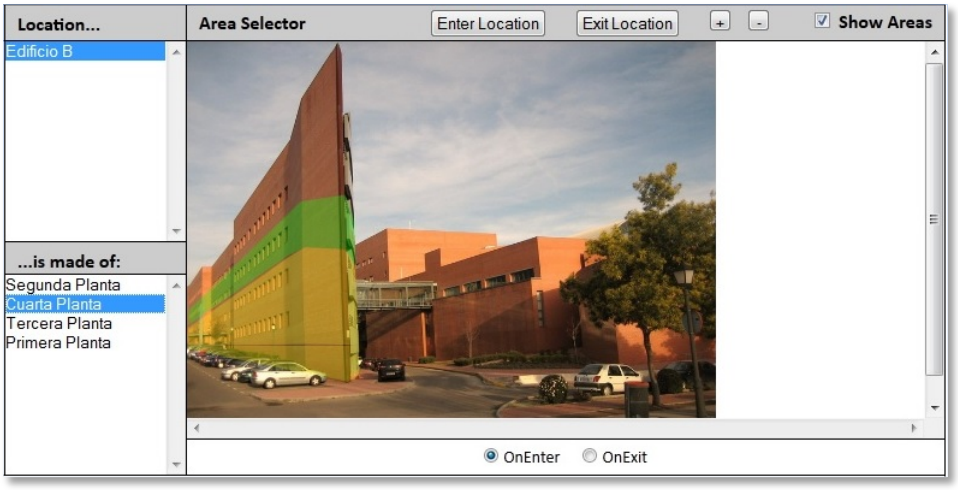

Figura 5-11: Interfaz ILB, selector de localización

Una vez determinada la entidad y la localización se especifica cuál de las acciones se debe comprobar mediante el selector radial *OnEnter/OnExit* según se refiera a la entrada o salida respectivamente.

Como ayuda visual, se presenta (de ser posible) una representación de la localización seleccionada, así como el pseudocódigo del evento que se está generando.

5.2.2 Especificación de las condiciones y acciones

El proceso de añadido de condiciones y acciones se lleva acabo empleando la misma forma de selección de localización usada en la generación de eventos para conseguir una lista de las entidades pertenecientes a dicho lugar y de esta forma escoger la entidad sobre cuyas capacidades se van a operar.

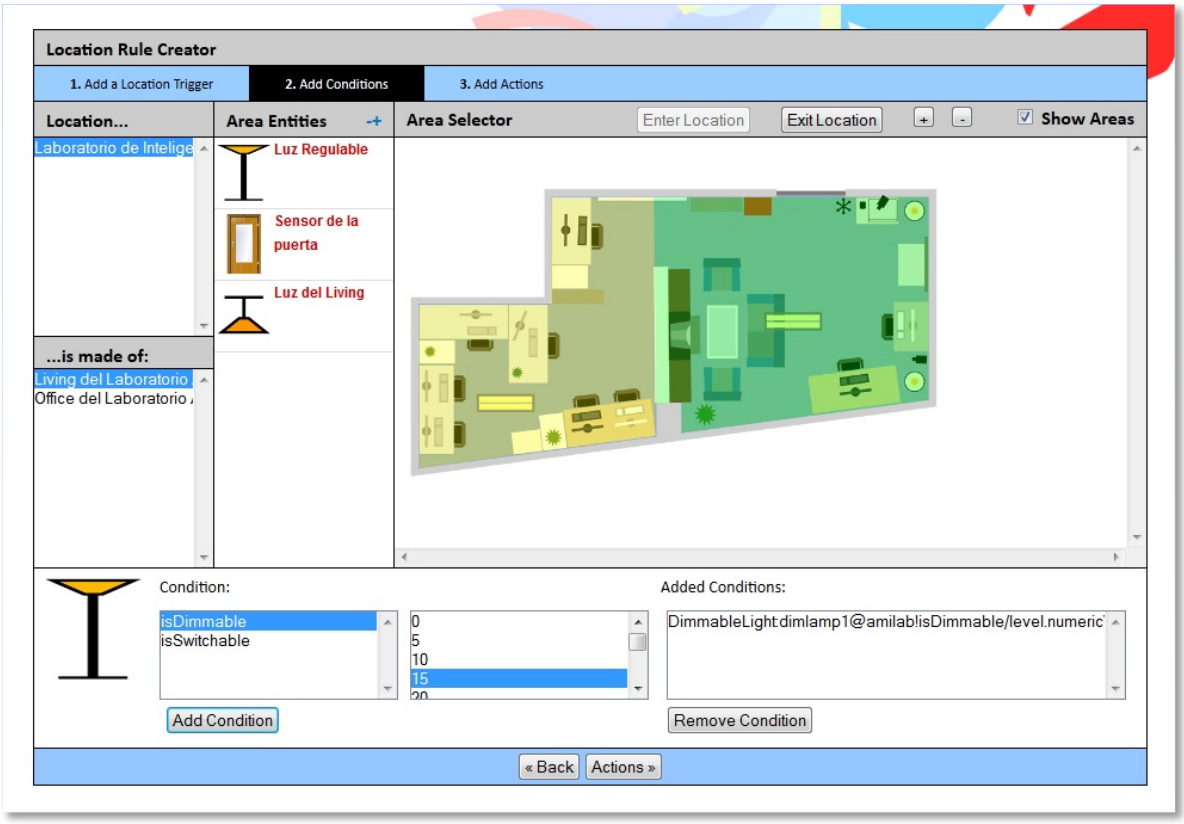

Figura 5-12: Interfaz ILB, proceso de creación de condiciones

Una vez seleccionada la entidad en la lista de *Area Entities*, como se puede ver en la figura 5-12, se muestra en el panel inferior un selector con las capacidades de la entidad, además de los posibles valores a asignar, y se procede a añadir las condiciones o acciones mediante el botón apropiado.

Por último, finalizada la selección del evento, condiciones (opcional) y acciones, se regresa a la herramienta original de control de agentes generándose el código en la regla desde la cual se inició el proceso.

5.3 Demostradores

5.3.1 Programando preferencias

Para ilustrar el empleo de las herramientas definidas para la creación y administración de agentes y reglas se va a realizar un ejemplo descrito a continuación.

Se desea poner en funcionamiento un agente propio de nombre *NachoLuces* que controle el apagado de las luces cuando Nacho abandone el laboratorio. Como condición adicional se desea que la luz regulable del salón del laboratorio solo este apagada de estar apagada ya antes, en caso contrario se deberá reducir la intensidad de luz al nivel más bajo permitido.

Empezando por la pantalla inicial, tal y como se ha visto en la Figura 5-1, se quiere crear un nuevo agente, por ello se pulsa en el botón correspondiente *New*, dando lugar a un campo de edición de texto en donde se introduce el nombre deseado, en este caso, *NachoLuces*. Una vez creado el agente se selecciona en la lista de *inactive bundles* y se comienza el editado por medio del botón *Edit*.

Siguiendo el método empleado para la creación del agente, se crea una nueva regla mediante el botón de *New*, en esta caso no es interesante ponerle nombre a la regla por lo que no se introduce nada en el campo de texto ofrecido y recibe una nomenclatura por defecto con una numeración. La regla creada es seleccionada por defecto por lo que se puede ver su contenido en cuadro de texto pertinente, que en este caso es un simple *placeholder* comentado sin significado para el agente. Se procede a acceder a la creación de reglas por el botón de *Generate* a lado del selector de herramienta de editado (que en nuestro ejemplo contiene solo la herramienta de localización y por tanto no es necesario modificar).

Una vez en la herramienta de creación de reglas se procede a la creación del evento de la regla. Para ello se selecciona la entidad sujeto de la regla a crear de la lista ofrecida por el selector de *Entities* visto en la Figura 5-10, *Nacho*, y se navega hasta el laboratorio a través de la lista de localizaciones por medio del botón de *Enter Location* y el selector adyacente. Seleccionado el sujeto y la localización del evento se escoge la opción *OnExit,* ya que se

desea que la regla se compruebe cuando Nacho abandona el laboratorio, y se procede a la especificación de condiciones por medio del botón *Conditions* en la parte inferior.

Para las condiciones, se busca identificar la luz regulable del salón del laboratorio, para ello, se selecciona el salón en el selector *is made of* (ya que la localización actual es el laboratorio dado que fue seleccionado en el evento) y se busca el elemento apropiado en la lista dada de entidades, identificable tanto por su icono como por el nombre *Luz Regulable*. Ya seleccionado, se tiene una lista de las propiedades sobre las que se puede actuar, y de entre ellas se selecciona la propiedad que se refiere al estado de encendido o apagado de la luz. Esto, a su vez, permite escoger de entre los dos estados posibles de la propiedad, escogiéndose, por tanto, el estado de encendido, buscándose comprobar que la luz este encendida como condición. Añadiendo la condición definida a la lista por medio del botón *Add Condition*, se procede a la especificación de acciones pulsándose en *Actions*, de nuevo en el panel inferior.

Para la adición de condiciones se procede de manera idéntica a la definición de condiciones, en este caso seleccionando las lámparas restantes del laboratorio e indicando que se desea que su estado pase a ser el de apagado, con excepción de la luz *dimmer* al cual se especifica que su propiedad de *dim* tenga por valor numérico 5, que es el valor del paso más pequeño distinto de cero. Una vez finalizada la regla se pulsa en el botón de *Finish*, volviendo así de nuevo a la herramienta de gestión de agentes, en donde la regla creada (Figura 5-13) ahora ocupa el cuadro de texto esperado.

```
!Person:nacho@amilab!locatedAt/Room:lab_B403@amilab
  ::
DimmableLight:dimlamp1@amilab!isSwitchable/
status.binaryValue = 1
  =>
Light:lamp1@amilab!isSwitchable/status.binaryValue := 0
 &&
Light:lamp1@amilab!isSwitchable/status.binaryValue := 0
 &&
DimmableLight:dimlamp1@amilab!isDimmable/
level.numericValue = 5
 ;
```

Figura 5-13: Primera regla obtenida del ejemplo para *NachoLuces*

Hasta ahora se ha definido como se quiere que actúe el agente en caso de estar encendida la luz regulable, para definir el caso contrario es necesario crear otra regla. Para ello repetimos los pasos anteriores, creando una nueva regla, de nuevo sin ser necesario especificar un nombre, y editando su contenido por medio de la herramienta de creación de reglas por localización, obteniendo la regla dada en la Figura 5-14.

```
!Person:nacho@amilab!locatedAt/Room:lab_B403@amilab
  ::
DimmableLight:dimlamp1@amilab!isSwitchable/
status.binaryValue = 0
  =>
Light:lamp1@amilab!isSwitchable/status.binaryValue := 0
 &&
Light:lamp1@amilab!isSwitchable/status.binaryValue := 0
 ;
```

Figura 5-14: Segunda regla obtenida del ejemplo para *NachoLuces*

Para finalizar, se guardan los cambios por medio de botón apropiado, indicado por el icono de guardado, y se cierra la edición del agente. Ya cerrado, se selecciona el agente *NachoLuces* en el selector de *inactive bundles*, que de nuevo se encuentra activo, y se pulsa en la flecha de activado indicada en la Figura 5-3: ahora ya tenemos nuestro agente creado y activo en la pizarra con las reglas diseñadas en funcionamiento.

Como se ha visto, el proceso de creación de agentes y reglas es un proceso sencillo y guiado, cuya complejidad se encapsula en la interfaz de usuario, permitiendo un manejo fácil e intuitivo.

5.3.2 Programando jerarquía

Se ha mencionado antes que los agentes son, a su vez, también entidades de la pizarra. Esto permite la programación de agentes controladores del estado de otros, permitiendo así codificar una jerarquía de agentes. Para ilustrar este proceso se incluirá a continuación un ejemplo.

Se tiene por un lado el agente desarrollado en el apartado anterior, *NachoLuces*, que ahora contiene todas las reglas relacionadas con el control de luces del laboratorio relativo a la posición de Nacho. Por otro lado, tenemos un agente nuevo, *JaviLuces*, creado de la misma forma, que determina el control de las luces según la posición de Javi. Se desea que las luces del laboratorio actúen en función de la posición de Javi mientras que Javi esté en el laboratorio y que funcionen según la posición de Nacho siempre que Nacho se encuentre en el laboratorio y Javi esté ausente. Para ello se va a crear un agente, *ControlLuces*, que determine cuál de los agentes de control de luces está activado en qué situación.

Se empieza activando ambos agentes, *NachoLuces* y *JaviLuces*, mediante las flechas de activación demostradas en el ejemplo anterior. De esta forma, los agentes pasan a ser entidades existentes en la pizarra y se puede actuar sobre sus propiedades. Después, se crea el agente nuevo y le damos el nombre deseado, *ControlLuces*, y se procede a su editado tal y como se ha descrito anteriormente. Para codificar el comportamiento deseado, se generan cinco reglas, que si bien pueden ser creadas parcialmente por el editor de reglas de localización, requieren del editado manual por medio del editor inferior (Figura 5-8) debido a la falta de soporte de la herramienta sobre las entidades de agentes:

Primero, se crea la regla encargada de activar el estado del agente *JaviLuces* cuando entra Javi en el laboratorio y de desactivar el agente *NachoLuces,* cuyo contenido se muestra en la Figura 5-15.

```
Person:javi@amilab!locatedAt/Room:lab_B403@amilab
  ::
 =>
Agent:javiluces@amilab!status/status.binaryValue := 1
 &&
Agent:nacholuces@amilab!status/status.binaryValue := 0
 ;
```

Figura 5-15: Primera regla de *ControlLuces*

Siguiendo por el control en relación a la posición de Javi se crea la regla encargada de desactivar *JaviLuces* una vez abandona el laboratorio, visto en la Figura 5-16.

```
!Person:javi@amilab!locatedAt/Room:lab_B403@amilab
  ::
 =>
Agent:javiluces@amilab!status/status.binaryValue := 0
;
```

Figura 5-16: Segunda regla de *ControlLuces*

En este caso, es necesario especificar la activación de *NachoLuces* si éste permanece dentro del laboratorio durante el evento de salida de Javi. Para ello se crea la regla vista en la Figura 5-17.

```
!Person:javi@amilab!locatedAt/Room:lab_B403@amilab
  ::
Person:nacho@amilab!locatedAt/Room:lab_B403@amilab
 =>
Agent:nacholuces@amilab!status/status.binaryValue := 1
;
```

Figura 5-17: Tercera regla de *ControlLuces*

Por último, se generan las reglas que controlan el estado de los agentes en función de la posición de Nacho. Por un lado, la activación del agente *NachoLuces* si entra Nacho en el laboratorio y *JaviLuces* se encuentra desactivado, Figura 5-18, y por otro la desactivación de *NachoLuces* al producirse la salida de Nacho del laboratorio, Figura 5-19.

```
Person:nacho@amilab!locatedAt/Room:lab_B403@amilab
  ::
Agent:javiluces@amilab!status/status.binaryValue = 0
 =>
Agent:nacholuces@amilab!status/status.binaryValue := 1
;
```

Figura 5-18: Cuarta regla de *ControlLuces*

```
!Person:nacho@amilab!locatedAt/Room:lab_B403@amilab
  ::
 =>
Agent:nacholuces@amilab!status/status.binaryValue := 0
 ;
```

Figura 5-19: Quinta regla de *ControlLuces*

Una vez creadas las reglas deseadas se procede al guardado y el activado del agente *ControlLuces*, de la forma ya descrita. Con esto ya se tiene la jerarquía programada deseada.

En este caso se puede apreciar la complejidad de la programación involucrada, si bien el desarrollo de la lógica necesaria es sencilla, la falta de soporte por parte del editor de reglas de localización dificulta la tarea para el usuario final. Esto se puede solucionar mediante la mejor integración de las entidades de agentes con sus propiedades en la herramienta de edición por localización pero una mejor opción, debido al diseño modular de la herramienta principal, sería la creación de un editor específico para la programación jerárquica.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

El objetivo de este proyecto era realizar una herramienta sencilla de cara al usuario que le permitiese programar el comportamiento que éste desea del entorno. Como se ha podido ver, el diseño del sistema basándose en agentes resulta en una manera intuitiva de organizar dicho comportamiento mediante agrupaciones de preferencias, además de permitir una programación del entorno por jerarquías en donde estos diversos conjuntos de comportamientos son activados en función del contexto.

A su vez, las reglas empleadas por dichos agentes, siguiendo el formato ECA que, como se ha visto, resulta en una programación intuitiva para el usuario común, permiten separar la capa de bajo nivel que se compone de los dispositivos del entorno, proporcionando un lenguaje de alto nivel sencillo pero potente y de gran precisión frente a otras alternativas.

Además, debido a la simplicidad de la sintaxis de dichas reglas, existe la posibilidad de automatizar el proceso de su generación por medio de herramientas especializadas, como el **ILB** mostrado en los apartados anteriores, que permite crear reglas basadas en localización.

Para desarrollar la interfaz de usuario se ha optado por emplear un diseño web, entorno fácilmente accesible para una gran cantidad de usuarios, cuyo modelo cliente-servidor permite separar el comportamiento de la propia interfaz de usuario con el de las funciones facilitadas por el sistema. El uso de módulos separados para la creación de reglas permite la creación de vistas adaptadas a los casos específicos, a diferencia de editores más genéricos ya mostrados.

6.2 Trabajo futuro

Como se ha mencionado anteriormente, la herramienta de control de agentes permite la fácil integración de múltiples herramientas de generación de reglas, por tanto, la creación de dichas herramientas es una tarea sobre la cual se puede y debería seguir iterando. En concreto, sería interesante la adición de un módulo orientado a la programación jerárquica, que permitiese especificar reglas relacionadas con el control de agentes de una manera sencilla.

Adicionalmente, resultaría interesante mejorar el sistema de organización de los agentes, permitiendo al usuario final realizar agrupaciones, por ejemplo, bajo nombres de usuario, además de permitir el uso de dichos usuarios como entidades en las reglas, pudiéndose así crear agentes que controlen las preferencias de activado de otros conjuntos de agentes bajo cada usuario.

Una de las ventajas de realizar la interfaz de usuario por medio de un servicio web es la posibilidad de expandir la funcionalidad a otros dispositivos, por lo que una posible ampliación del proyecto sería adaptarse a plataformas móviles, más comunes a día de hoy. Como continuación sobre esta alternativa estaría el desarrollo de aplicaciones específicas para *smartphones* y *tablets* como interfaces alternativas.

Referencias

[1] Davidoff, S., Lee, M., Zimmerman, J., Dey, A.: \Socially-aware requirements for a smart home"; Procs of the International Symposium on Intelligent Environments; 41{44; 2006.

[2] Matilda Smart House - Smart Space Deployments, Mobile & Pervasive Computing Research, University of Florida. http:// www.icta.ufl.edu/gt.htm#2

[3] A. Helal, C. Giraldo, Y. Kaddourah, C. Lee, H. Zabadani, and W. Mann, "Smart Phone Based Cognitive Assistant", November 2003.

[4] The Gator Tech Smart House - Smart Space Deployments, Mobile & Pervasive Computing Research, University of Florida. http:// http://www.icta.ufl.edu/gt.htm#1

[5] A. Helal, W. Mann, H. Elzabadani, J. King, Y. Kaddourah and E. Jansen, "Gator Tech Smart House: A Programmable Pervasive Space", IEEE Computer magazine, March 2005, pp 64-74.

[6] Amigo Project. http://www.hitech-projects.com/euprojects/amigo/amigo.htm

[7] SMARTLAB - Entorno de Trabajo Inteligente Colaborativo y Programable, 2007. http://www.tecnologico.deusto.es/projects/smartlab/

[8] Aware Home Research Initiative. http://www.awarehome.gatech.edu/

[9] I. Essa, "Ubiquitous Sensing for Smart and Aware Environments: Technologies towards the building of an Aware Home", July 1999.

[10] AmILab - ambient intelligence laboratory. http://amilab.ii.uam.es/

[11] M. García-Herranz, P. A. Haya, A. Esquivel, G. Montoro and X. Alamán, "Easing the smart home: Semiautomatic adaptation in perceptive environments" Journal of Universal Computer Science, vol. 14, no. 9, pp. 1529–1544, may 2008.

[12] Manuel García-Herranz del Olmo, Xavier Alamán Roldán, Pablo A. Haya Coll, Pablo Martín, "Easing the Smart Home: augmenting devices and defining scenarios" 2nd International Symposium on Ubiquitous Computing&Ambient, Thomson, editorial, 2007. pp. 67-74, ISBN: 9788497326056. Best Paper Award.

[13] Pons Tomás, P. (2012), "DaFRULE: un modelo de reglas enriquecido mediante flujos de datos para la definición visual de comportamientos en entornos reactivos", Jaén Martínez, FJ. dir. ; Catalá Bolós, A. dir. 88 p.

[14] Weis, T., Handte, M., Knoll, M., Becker, C., "Customizable Pervasive Applications" In Proc. of PerCom'06, pp. 239–244 (2006).

[15] Anind K. Dey, Raffay Hamid, Chris Beckmann, Ian Li, Daniel Hsu, "a CAPpella: Programming by Demonstration of Context-Aware Applications", CHI 2004.

[16] García-Herranz, M., Haya, P.A., Alamán, X. "Towards an ubiquitous end–user programming system for Smart Spaces", accepted for publication in Journal of Universal Computer Science (JUCS). 2010.

[17] García Herranz, M., Thesis: "Easing the Smart Home: a rule–based language and multi–agent structure for end user development in Intelligent Environments", 2009.

PRESUPUESTO

1) Ejecución Material

- Compra de ordenador personal (Software incluido)..... ..... 2.000 €
- Alquiler de impresora láser durante 6 meses..... 50 €
- Material de oficina..... 150 €
- Total de ejecución material ..... 2.200 €

2) Gastos generales

- 21 % sobre Ejecución Material ..... 462 €

3) Beneficio Industrial

- 6 % sobre Ejecución Material ..... 132 €

4) Honorarios Proyecto

- 640 horas a 15 € / hora ..... 9600 €

5) Material fungible

- Gastos de impresión ..... 60 €
- Encuadernación ..... 200 €

6) Subtotal del presupuesto

- Subtotal Presupuesto ..... 12060 €

7) I.V.A. aplicable

- 21% Subtotal Presupuesto..... 2532,6 €

8) Total presupuesto

- Total Presupuesto ..... 14592,6 €

Madrid, Abril de 2013

El Ingeniero Jefe de Proyecto

Fdo.: Peter Trullenque Eriksson
Ingeniero Superior de Telecomunicación

PLIEGO DE CONDICIONES

Este documento contiene las condiciones legales que guiarán la realización, en este proyecto, de un sistema de Personalización de Entornos de Inteligencia Ambiental. En lo que sigue, se supondrá que el proyecto ha sido encargado por una empresa cliente a una empresa consultora con la finalidad de realizar dicho sistema. Dicha empresa ha debido desarrollar una línea de investigación con objeto de elaborar el proyecto. Esta línea de investigación, junto con el posterior desarrollo de los programas está amparada por las condiciones particulares del siguiente pliego.

Supuesto que la utilización industrial de los métodos recogidos en el presente proyecto ha sido decidida por parte de la empresa cliente o de otras, la obra a realizar se regulará por las siguientes:

Condiciones generales

1. La modalidad de contratación será el concurso. La adjudicación se hará, por tanto, a la proposición más favorable sin atender exclusivamente al valor económico, dependiendo de las mayores garantías ofrecidas. La empresa que somete el proyecto a concurso se reserva el derecho a declararlo desierto.

2. El montaje y mecanización completa de los equipos que intervengan será realizado totalmente por la empresa licitadora.

3. En la oferta, se hará constar el precio total por el que se compromete a realizar la obra y el tanto por ciento de baja que supone este precio en relación con un importe límite si este se hubiera fijado.

4. La obra se realizará bajo la dirección técnica de un Ingeniero Superior de Telecomunicación, auxiliado por el número de Ingenieros Técnicos y Programadores que se estime preciso para el desarrollo de la misma.

5. Aparte del Ingeniero Director, el contratista tendrá derecho a contratar al resto del personal, pudiendo ceder esta prerrogativa a favor del Ingeniero Director, quien no estará obligado a aceptarla.

6. El contratista tiene derecho a sacar copias a su costa de los planos, pliego de condiciones y presupuestos. El Ingeniero autor del proyecto autorizará con su firma las copias solicitadas por el contratista después de confrontarlas.

7. Se abonará al contratista la obra que realmente ejecute con sujeción al proyecto que sirvió de base para la contratación, a las modificaciones autorizadas por la superioridad o a las órdenes que con arreglo a sus facultades le hayan comunicado por escrito al Ingeniero Director de obras siempre que dicha obra se haya ajustado a los preceptos de los pliegos de condiciones, con arreglo a los cuales, se harán las modificaciones y la valoración de las diversas unidades sin que el importe total pueda exceder de los presupuestos aprobados. Por consiguiente, el número de unidades que se consignan en el proyecto o en el presupuesto, no podrá servirle de fundamento para entablar reclamaciones de ninguna clase, salvo en los casos de rescisión.

8. Tanto en las certificaciones de obras como en la liquidación final, se abonarán los trabajos realizados por el contratista a los precios de ejecución material que figuran en el presupuesto para cada unidad de la obra.

9. Si excepcionalmente se hubiera ejecutado algún trabajo que no se ajustase a las condiciones de la contrata pero que sin embargo es admisible a juicio del Ingeniero Director de obras, se dará conocimiento a la Dirección, proponiendo a la vez la rebaja de precios que el Ingeniero estime justa y si la Dirección resolviera aceptar la obra, quedará el contratista obligado a conformarse con la rebaja acordada.

10. Cuando se juzgue necesario emplear materiales o ejecutar obras que no figuren en el presupuesto de la contrata, se evaluará su importe a los precios asignados a otras obras o materiales análogos si los hubiere y cuando no, se discutirán entre el Ingeniero Director y el contratista, sometiéndolos a la aprobación de la Dirección. Los nuevos precios convenidos por uno u otro procedimiento, se sujetarán siempre al establecido en el punto anterior.

11. Cuando el contratista, con autorización del Ingeniero Director de obras, emplee materiales de calidad más elevada o de mayores dimensiones de lo estipulado en el proyecto, o sustituya una clase de fabricación por otra que tenga asignado mayor precio o ejecute con mayores dimensiones cualquier otra parte de las obras, o en general, introduzca en ellas cualquier modificación que sea beneficiosa a juicio del Ingeniero Director de obras, no tendrá derecho sin embargo, sino a lo que le correspondería si hubiera realizado la obra con estricta sujeción a lo proyectado y contratado.

12. Las cantidades calculadas para obras accesorias, aunque figuren por partida alzada en el presupuesto final (general), no serán abonadas sino a los precios de la contrata, según las condiciones de la misma y los proyectos particulares que para ellas se formen, o en su defecto, por lo que resulte de su medición final.

13. El contratista queda obligado a abonar al Ingeniero autor del proyecto y director de obras así como a los Ingenieros Técnicos, el importe de sus respectivos honorarios facultativos por formación del proyecto, dirección técnica y administración en su caso, con arreglo a las tarifas y honorarios vigentes.

14. Concluida la ejecución de la obra, será reconocida por el Ingeniero Director que a tal efecto designe la empresa.

15. La garantía definitiva será del 4% del presupuesto y la provisional del 2%.

16. La forma de pago será por certificaciones mensuales de la obra ejecutada, de acuerdo con los precios del presupuesto, deducida la baja si la hubiera.

17. La fecha de comienzo de las obras será a partir de los 15 días naturales del replanteo oficial de las mismas y la definitiva, al año de haber ejecutado la provisional, procediéndose si no existe reclamación alguna, a la reclamación de la fianza.

18. Si el contratista al efectuar el replanteo, observase algún error en el proyecto, deberá comunicarlo en el plazo de quince días al Ingeniero Director de obras, pues transcurrido ese plazo será responsable de la exactitud del proyecto.

19. El contratista está obligado a designar una persona responsable que se entenderá con el Ingeniero Director de obras, o con el delegado que éste designe, para todo relacionado con ella. Al ser el Ingeniero Director de obras el que interpreta el proyecto, el contratista deberá consultarle cualquier duda que surja en su realización.

20. Durante la realización de la obra, se girarán visitas de inspección por personal facultativo de la empresa cliente, para hacer las comprobaciones que se crean oportunas. Es obligación del contratista, la conservación de la obra ya ejecutada hasta la recepción de la misma,

por lo que el deterioro parcial o total de ella, aunque sea por agentes atmosféricos u otras causas, deberá ser reparado o reconstruido por su cuenta.

21. El contratista, deberá realizar la obra en el plazo mencionado a partir de la fecha del contrato, incurriendo en multa, por retraso de la ejecución siempre que éste no sea debido a causas de fuerza mayor. A la terminación de la obra, se hará una recepción provisional previo reconocimiento y examen por la dirección técnica, el depositario de efectos, el interventor y el jefe de servicio o un representante, estampando su conformidad el contratista.

22. Hecha la recepción provisional, se certificará al contratista el resto de la obra, reservándose la administración el importe de los gastos de conservación de la misma hasta su recepción definitiva y la fianza durante el tiempo señalado como plazo de garantía. La recepción definitiva se hará en las mismas condiciones que la provisional, extendiéndose el acta correspondiente. El Director Técnico propondrá a la Junta Económica la devolución de la fianza al contratista de acuerdo con las condiciones económicas legales establecidas.

23. Las tarifas para la determinación de honorarios, reguladas por orden de la Presidencia del Gobierno el 19 de Octubre de 1961, se aplicarán sobre el denominado en la actualidad "Presupuesto de Ejecución de Contrata" y anteriormente llamado "Presupuesto de Ejecución Material" que hoy designa otro concepto.

Condiciones particulares

La empresa consultora, que ha desarrollado el presente proyecto, lo entregará a la empresa cliente bajo las condiciones generales ya formuladas, debiendo añadirse las siguientes condiciones particulares:

1. La propiedad intelectual de los procesos descritos y analizados en el presente trabajo, pertenece por entero a la empresa consultora representada por el Ingeniero Director del Proyecto.

2. La empresa consultora se reserva el derecho a la utilización total o parcial de los resultados de la investigación realizada para desarrollar el siguiente proyecto, bien para su publicación o bien para su uso en trabajos o proyectos posteriores, para la misma empresa cliente o para otra.

3. Cualquier tipo de reproducción aparte de las reseñadas en las condiciones generales, bien sea para uso particular de la empresa cliente, o para cualquier otra aplicación, contará con autorización expresa y por escrito del Ingeniero Director del Proyecto, que actuará en representación de la empresa consultora.

4. En la autorización se ha de hacer constar la aplicación a que se destinan sus reproducciones así como su cantidad.

5. En todas las reproducciones se indicará su procedencia, explicitando el nombre del proyecto, nombre del Ingeniero Director y de la empresa consultora.

6. Si el proyecto pasa la etapa de desarrollo, cualquier modificación que se realice sobre él, deberá ser notificada al Ingeniero Director del Proyecto y a criterio de éste, la empresa consultora decidirá aceptar o no la modificación propuesta.

7. Si la modificación se acepta, la empresa consultora se hará responsable al mismo nivel que el proyecto inicial del que resulta el añadirla.

8. Si la modificación no es aceptada, por el contrario, la empresa consultora declinará toda responsabilidad que se derive de la aplicación o influencia de la misma.

9. Si la empresa cliente decide desarrollar industrialmente uno o varios productos en los que resulte parcial o totalmente aplicable el estudio de este proyecto, deberá comunicarlo a la empresa consultora.

10. La empresa consultora no se responsabiliza de los efectos laterales que se puedan producir en el momento en que se utilice la herramienta objeto del presente proyecto para la realización de otras aplicaciones.

11. La empresa consultora tendrá prioridad respecto a otras en la elaboración de los proyectos auxiliares que fuese necesario desarrollar para dicha aplicación industrial, siempre que no haga explícita renuncia a este hecho. En este caso, deberá autorizar expresamente los proyectos presentados por otros.

12. El Ingeniero Director del presente proyecto, será el responsable de la dirección de la aplicación industrial siempre que la empresa consultora lo estime oportuno. En caso contrario, la persona designada deberá contar con la autorización del mismo, quien delegará en él las responsabilidades que ostente.

Anexo

Código fuente de las clases y métodos principales

AgentManager.java

```
  public String renameAgent(String info){
      String nameTaken = "false";
      String target = info.split(";")[0];
      String newName = info.split(";")[1];

      File folder = new File(agentPath+"\\rules");
      File[] listOfFiles = folder.listFiles();
      for (int i = 0; i < listOfFiles.length; i++) {
          if (listOfFiles[i].isFile()) {
              String agentName = listOfFiles[i].getName().replace(".dat", "");
              if (newName.equalsIgnoreCase(agentName)){
                nameTaken = "true";
              }
          }
      }

      if(nameTaken.equalsIgnoreCase("false") && newName.length()!=0){
          _logger.info("renaming "+target+" to "+newName);
          File targetFile = new File(agentPath+"\\rules\\"+target+".dat");
          targetFile.renameTo(new File(agentPath+"\\rules\\"+newName+".dat"));
      }

      return nameTaken;
  }

  public String newAgent(String name){
      String nameTaken = "false";

      File folder = new File(agentPath+"\\rules");
      File[] listOfFiles = folder.listFiles();
      for (int i = 0; i < listOfFiles.length; i++) {
          if (listOfFiles[i].isFile()) {
              String agentName = listOfFiles[i].getName().replace(".dat", "");
              if (name.equalsIgnoreCase(agentName)){
                nameTaken = "true";
              }
          }
      }

      if(nameTaken.equalsIgnoreCase("false") && name.length()!=0){
          try{
              FileWriter file = new FileWriter(new
File(agentPath+"\\rules\\"+name+".dat"));
              BufferedWriter out = new BufferedWriter(file);
              out.write("#placeholder\n");
              out.close();
          }catch (IOException ex) {
```

```
                _logger.info("Error when creating rules file: " + name);
            }
        }

        return nameTaken;
    }

    public void removeAgent(String info){
        String[] names = info.split(";");
        for(String name : names){
            File targetFile = new File(agentPath+"\\rules\\"+name+".dat");
            targetFile.delete();
        }
    }

    public void shutdownAgents(){
        int i;
        Iterator it = agentProcessMap.values().iterator();
        for(i=0; i<agentProcessMap.size();i++){
            _logger.info("Sending console quit command to agent #"+i);
            String line;
            Process current = (Process)it.next();
            OutputStream stdin = current.getOutputStream();
            InputStream stderr = current.getErrorStream();
            InputStream stdout = current.getInputStream();

            line = "q" + "\n";
            try {
                stdin.write(line.getBytes());
                stdin.flush();
                stdin.close();
                stderr.close();
                stdout.close();
            } catch (IOException ex) {}
        }
    }

    public String getBundles() {
        File folder = new File(agentPath+"\\rules");
        File[] listOfFiles = folder.listFiles();
        StringBuffer sb = new StringBuffer();
        for (int i = 0; i < listOfFiles.length; i++) {
            if (listOfFiles[i].isFile()) {
                boolean active = false;
                String agentName = listOfFiles[i].getName().replace(".dat", "");
for (String s : agentProcessMap.keySet()) {
                    if (agentName.equalsIgnoreCase(s)){
                        active = true;
                    }
                }
                if (!active){
                    sb.append("<option value=\"" + agentName + "\">" + agentName +
"</option>");
                }
            }
        }
```

```
        return sb.toString();
    }

    public String getActiveBundles() {

        StringBuffer sb = new StringBuffer();
        for (String s : agentProcessMap.keySet()) {
            sb.append("<option value=\"" + s + "\">" + s + "</option>");
        }
        return sb.toString();
    }

    public void loadRulesFile(String name) {
        try {
            Map<String, String> ruleMap = new TreeMap<String, String>();
            FileReader file = new FileReader(new
File(agentPath+"\\rules\\"+name+".dat"));
            BufferedReader RulesBuffer = new BufferedReader(file);
            String strRule = RulesBuffer.readLine();
            String ruleName="";
            int index = 1;
            while (strRule != null) {
                strRule = strRule.trim();
                if (!strRule.equals("")) {
                    if (strRule.charAt(0) == '#') {
                        if(strRule.contains("#name:")){
                            ruleName=strRule.replace("#name:", "");
                        }
                    }else{
                        boolean taken = false;
                        for (String s : ruleMap.keySet()) {
                            if (ruleName.equalsIgnoreCase(s)){
                              taken = true;
                            }
                        }
                        do{
                            if(ruleName=="" || taken){
                                ruleName="Rule "+index;
                                index++;
                                taken = false;
                            }
                            for (String s : ruleMap.keySet()) {
                                if (ruleName.equalsIgnoreCase(s)){
                                  taken = true;
                                }
                            }
                        }while(taken==true);
                        ruleMap.put(ruleName, strRule);
                        ruleName="";
                    }
                }
                strRule = RulesBuffer.readLine();
            }
            file.close();
            editMap.put(name, ruleMap);
            _logger.info(name + "'s rules file loaded.");
        } catch (IOException e) {
```

```
            _logger.info("Error when loading rules file: " + name);
        }
    }

    public void closeRulesFile(String name) {
        editMap.remove(name);
        _logger.info(name + "'s rules file closed.");
    }

    public void saveRulesFile(String name) {
        try{
            Map<String, String> ruleMap = new TreeMap<String, String>();
            ruleMap = editMap.get(name);
            FileWriter file = new FileWriter(new
File(agentPath+"\\rules\\"+name+".dat"));
            BufferedWriter out = new BufferedWriter(file);
            for (String s : ruleMap.keySet()) {
                out.write("#name:"+s+"\n"+ruleMap.get(s)+"\n");
            }
            out.close();
        }catch (IOException ex) {
            _logger.info("Error when saving rules file: " + name);
        }
    }

    public String getRulesFromMap(String name) {
        StringBuffer sb = new StringBuffer();
        for (String s : editMap.get(name).keySet()) {
            sb.append("<option value=\"" + s + "\">" + s + "</option>");
        }
        _logger.info(name + "'s rfm: "+sb);
        return sb.toString();
    }

    public String getRule(String info) {
        String[] data = info.split(";");
        String rule = editMap.get(data[0]).get(data[1]);
        _logger.info("getRule: "+info+";"+rule);
        return rule;
    }

    public String newRule(String info) {
        String[] data = info.split(";");
        int index = 1;
        String ruleName;
        Map<String, String> ruleMap = new TreeMap<String, String>();
        ruleMap = editMap.get(data[0]);
        if(data.length==1){
            ruleName = "(New Rule) "+index;
            index++;
        }else{
            ruleName = data[1];
        }
        boolean taken = false;
        for (String s : ruleMap.keySet()) {
            if (ruleName.equalsIgnoreCase(s)){
                taken = true;
```

```
            }
        }
        while(taken==true){
            ruleName="(New Rule) "+index;
            index++;
            taken = false;
            for (String s : ruleMap.keySet()) {
                if (ruleName.equalsIgnoreCase(s)){
                    taken = true;
                }
            }
        }

        setRule(data[0]+";"+ruleName+";#<replace with a valid rule>");
        return ruleName;
    }

    public void removeRule(String info) {
        String[] data = info.split(";");
        for(int i = 1; i<data.length; i++){
            Map<String, String> ruleMap = new TreeMap<String, String>();
            ruleMap = editMap.get(data[0]);
            ruleMap.remove(data[i]);
            editMap.put(data[0], ruleMap);
        }
    }

    public String checkRuleName (String info){
        String[] data = info.split(";");
        for (String s : editMap.get(data[0]).keySet()){
            if (data[1].equalsIgnoreCase(s)){
                return "false";
            }
        }
        return "true";
    }

    public String renameRule (String info) {
        String[] data = info.split(";");
        Map<String, String> ruleMap = new TreeMap<String, String>();
        ruleMap = editMap.get(data[0]);
        String newRule = editMap.get(data[0]).get(data[1]);
        ruleMap.put(data[2], newRule);
        ruleMap.remove(data[1]);
        editMap.put(data[0], ruleMap);

        return getRulesFromMap(data[0]);
    }

    public void setRule(String info) {
        _logger.info("setRule: "+info);
        String[] data = info.split(";");
        String newRule = data[2].replace("[and]", "&")+";";
        Map<String, String> ruleMap = new TreeMap<String, String>();
        ruleMap = editMap.get(data[0]);
        ruleMap.put(data[1], newRule);
        editMap.put(data[0], ruleMap);
```

```
}

public void moveRule(String info) {
   String[] data = info.split(";");
   loadRulesFile(data[1]);
   int index = 1;
   for(int i = 2; i<data.length; i++){
      Map<String, String> ruleMap = new TreeMap<String, String>();
      ruleMap = editMap.get(data[1]);
      String newRule = editMap.get(data[0]).get(data[i]);
      String ruleName = data[i];
      boolean taken = false;
      for (String s : ruleMap.keySet()) {
         if (ruleName.equalsIgnoreCase(s)){
         taken = true;
         }
      }
      while(taken==true){
         ruleName="(Moved Rule) "+index;
         index++;
         taken = false;
         for (String s : ruleMap.keySet()) {
            if (ruleName.equalsIgnoreCase(s)){
               taken = true;
            }
         }
      }
      ruleMap.put(ruleName, newRule);
      editMap.put(data[1], ruleMap);

      ruleMap = editMap.get(data[0]);
      ruleMap.remove(data[i]);
      editMap.put(data[0], ruleMap);
   }
   saveRulesFile(data[0]);
   saveRulesFile(data[1]);
   closeRulesFile(data[1]);
   _logger.info("moveRule: "+info);
}

public String checkAgent(String info) {
   String[] names = info.split(";");

   String active = "false";
   String edit = "false";

   for(String name : names){
      for (String s : agentProcessMap.keySet()) {
         if (name.equalsIgnoreCase(s)){
            active = "true";
         }
      }

      for (String s : editMap.keySet()) {
         if (name.equalsIgnoreCase(s)){
            edit = "true";
         }
```

```
            }
        }
        if(freePorts.size()+(maxPortOffset-portOffset)<names.length){
            _logger.info("Free ports = false; Bundles already active = "+active+";
Bundles already editing = "+edit);
            return "false;"+active+";"+edit;
        }else{
            _logger.info("Free ports = true; Bundles already active = "+active+";
Bundles already editing = "+edit);
            return "true;"+active+";"+edit;
        }
    }

        public void startAgent(String info) throws IOException {
        String[] names = info.split(";");
        for(String name : names){
            boolean active = false;
            for (String s : agentProcessMap.keySet()) {
                if (name.equalsIgnoreCase(s)){
                    active = true;
                }
            }
            if(!active){
                String agentPort= new String();
                if(!freePorts.isEmpty()){
                    agentPort = freePorts.get(0);
                    freePorts.remove(0);
                }else{
                    agentPort = String.valueOf(minPort+portOffset);
                    portOffset=portOffset+1;
                }
                String rulesFile = "rules/"+name;
                ProcessBuilder pb = new ProcessBuilder("cmd", "/C java -cp
c:\\lib\\Repository.jar;c:\\lib\\ant.jar;c:\\lib\\antlr.jar;c:\\lib\\bbcommon.jar;c:\\lib\\
bbnamesolver.jar;c:\\lib\\bbrepository.jar;c:\\lib\\bbauthorization.jar;c:\\lib\\bbsche
ma.jar;c:\\lib\\Ice.jar;c:\\lib\\javabase64-
1.2.jar;c:\\lib\\libutil.jar;c:\\lib\\bbevents.jar;c:\\lib\\libbb.jar;.\\agentesBB2.jar;
agent.Agente "+name+" "+agentPort+" "+rulesFile);
                File dir = new File(agentPath);
                pb.directory(dir);
                agentProcessMap.put(name, pb.start());
                agentPortMap.put(name, agentPort);
            }else{
                _logger.info(name+" already active.");
            }
        }

    }

    public void stopAgent(String info) throws IOException {
        String[] names = info.split(";");
        for(String name : names){
            boolean active = false;
            for (String s : agentProcessMap.keySet()) {
                if (name.equalsIgnoreCase(s)){
                    active = true;
                }
```

```
        }
        if(active){
            _logger.info("Sending console quit command to "+name);
            String line;
            OutputStream stdin = agentProcessMap.get(name).getOutputStream();
            InputStream stderr = agentProcessMap.get(name).getErrorStream();
            InputStream stdout = agentProcessMap.get(name).getInputStream();

            line = "q" + "\n";
            stdin.write(line.getBytes() );
            stdin.flush();

            stdin.close();
            stderr.close();
            stdout.close();

            freePorts.add(agentPortMap.get(name));

            agentProcessMap.remove(name);
            agentPortMap.remove(name);
        }else{
            _logger.info(name+" already inactive.");
        }
    }
}
```

BBInfo.java

```
public String getEntityProperties(String path) {
    try {
        String st[] = _bb.getEntityPropertiesNamesFromPath(path);
        if (st != null && st.length != 0) {
            StringBuffer sb = new StringBuffer();
            int i = 0;
            for (String s : st) {
                if (i == 0) {
                    sb.append(s);
                } else {
                    sb.append(";" + s);
                }
                i++;
            }

            return sb.toString();
        } else {
            return null;
        }
    } catch (BlackBoardException ex) {
        return null;
    }

}
```

```
    public String getEntityCapabilities(String path) {
        try {
            String st[] = _bb.getCapabilitiesNamesFromPath(path);
            if (st != null && st.length != 0) {
                StringBuffer sb = new StringBuffer();
                for (String s : st) {
                    sb.append("<option value=\"" + s + "\">" + s + "</option>");
                }
                return sb.toString();
            } else {
                return null;
            }
        } catch (BlackBoardException ex) {
            return null;
        }

    }

    public String getEntityCapabilityValues(String path) {
        try {
            _logger.info("Get Entity Capability Values from:"+path);
            StringBuffer sb = new StringBuffer();
            CapabilityPrx Cap = _bb.getCapabilityFromPath(path);
            if (Cap != null) {
                Map<String,PropertyPrx> map  = Cap.getProperties(null);
                if (map.values()!=null){
                    for(PropertyPrx prop:map.values()){
                            Map<String,PropertyPrx>[] map2s  =
prop.getPropertiesFromStruct(null);
                        if (map2s!=null){
                            for(Map<String,PropertyPrx> map2:map2s){
                              int isNumeric = 0, min=0, max=0, step=1;
                              for(PropertyPrx prop2:map2.values()){
                                 String Propname = prop2.getName(null);
                                 if (Propname.equals("binaryValue")){
                                    String[] vals =
_bb.getCapabilityPropertyValuesFromPath(path+"/"+prop.getName(null)+".represent
ation.iText.iTextSpanish.texto");
                                    sb.append("<option
value=\""+prop.getName(null)+"."+Propname+"=0"+"\">"+ vals[0] +"</option>");
                                    sb.append("<option
value=\""+prop.getName(null)+"."+Propname+"=1"+"\">"+ vals[1] +"</option>");
                                 }
                                 if (Propname.equals("numericValue")){
                                    isNumeric = 1;
                                 }
                                 if (Propname.equals("minValue")){
                                    min =
Integer.parseInt(prop2.getDefaultValueAsString(null));
                                 }
                                 if (Propname.equals("maxValue")){
                                    max =
Integer.parseInt(prop2.getDefaultValueAsString(null));
                                 }
                                 if (Propname.equals("step")){
                                    step =
Integer.parseInt(prop2.getDefaultValueAsString(null));
```

```
                        }
                    }
                    if(isNumeric == 1){
                        for(int i=min;i<=max;i=i+step){
                            sb.append("<option
value=\""+prop.getName(null)+".numericValue="+i+"\">"+ i +"</option>");
                        }
                    }
                }
            }
        }
        _logger.info("Results in:"+sb.toString());
        return sb.toString();
        }else {
            return null;
        }
    } catch (BlackBoardException ex) {
        return null;
    }

}

public String getEntityRelations(String path) {
    try {
        String st[] = _bb.getRelationsNamesFromPath(path);
        if (st != null && st.length != 0) {
            StringBuffer sb = new StringBuffer();
            int i = 0;
            for (String s : st) {
                if (i == 0) {
                    sb.append(s);
                } else {
                    sb.append(";" + s);
                }
                i++;
            }
            return sb.toString();
        } else {
            return null;
        }
    } catch (BlackBoardException ex) {
        return null;
    } }

public String getRangeEntities() {
    try {
        _logger.info("T_Entities step 1.");
        Map<String, EntityPrx> ents =
_bb.getEntitiesFromPath("Person:*@amilab");
        _logger.info("T_Entities step 2.");
        StringBuffer sb = new StringBuffer();
        for (String s : ents.keySet()) {
                String name =
_bb.getEntityPropertyValueFromPath(s+"!iName.iNameSpanish");
                sb.append("<option value=\"" + s + "\">" + name + "</option>");
        }
```

```
            return sb.toString();
        } catch (BlackBoardException ex) {
            return null;
        }

    }

    public String getEntities() {
        try {
            Map<String, EntityPrx> ents = _bb.getEntities("amilab");
            StringBuffer sb = new StringBuffer();
            int i = 0;
            for (EntityPrx e : ents.values()) {
                if (i == 0) {
                    sb.append("<option selected title=\"" + e.getFullName(null) + "\">" +
e.getFullName(null) + "</option>");
                } else {
                    sb.append("<option title=\"" + e.getFullName(null) + "\">" +
e.getFullName(null) + "</option>");
                }
                i++;
            }
            return sb.toString();
        } catch (BlackBoardException ex) {
            return null;
        }
    }

    public String getFirstEntities() {
        try {
            Map<String, EntityPrx> ents = _bb.getEntitiesFilterByType("Building",
"amilab");
            StringBuffer sb = new StringBuffer();
            for (String s : ents.keySet()) {
                String name =
_bb.getEntityPropertyValueFromPath(s+"!iName.iNameSpanish");
                sb.append("<option value=\"" + s + "\">" + name + "</option>");
            }
            return sb.toString();
        } catch (BlackBoardException ex) {
            return null;
        }

    }

    public String getLocationComposition(String path) {
        _logger.info("path="+path);
        String entData[] = Checkers.getEntityDataFromString(path, 0);
        try {

            Map<String, RelationPrx> rels = _bb.getRelationsFilterByName(entData[0],
entData[1], entData[2], "isComposedOf");
            StringBuffer sb = new StringBuffer();
            int i = 0;
            for (String s : rels.keySet()) {
                String name =
_bb.getEntityPropertyValueFromPath(s.split("/")[1]+"!iName.iNameSpanish");
```

```
                sb.append(i == 0 ? (s.split("/")[1] + "," + name) : (";" + s.split("/")[1]
+ "," + name));
                i++;
            }
            sb.append("?" + getComposes(path));
            _logger.info("sb="+sb.toString());
            return sb.toString();
        } catch (BlackBoardException ex) {
            return null;
        }
    }

    public String getIsComposedOf(String path) {
        String entData[] = Checkers.getEntityDataFromString(path, 0);
        try {
            Map<String, RelationPrx> rels = _bb.getRelationsFilterByName(entData[0],
entData[1], entData[2], "isComposedOf");
            if (rels != null && rels.size() != 0) {
                return "true";
            } else {
                return "false";
            }

        } catch (BlackBoardException ex) {
            return null;
        }
    }

    public String exitLocation(String path) {
        String entData[] = Checkers.getEntityDataFromString(path, 0);
        try {
            Map<String, RelationPrx> rels = _bb.getRelationsFilterByName(entData[0],
entData[1], entData[2], "composes");
            if (rels != null && rels.size() != 0) {
                StringBuffer sb = new StringBuffer();
                for (String s : rels.keySet()) {
                    String name =
_bb.getEntityPropertyValueFromPath(s.split("/")[1]+"!iName.iNameSpanish");
                    sb.append(s.split("/")[1] + "," + name);
                }
                _logger.info(sb.toString());
                return sb.toString();
            } else {
                return "error";
            }

        } catch (BlackBoardException ex) {
            return null;
        }
    }

    public String getAreaEntities(String path) {
        try {
            Map<String, EntityPrx> ents = _bb.getEntitiesFilterByRelation("locatedAt/" +
path, "amilab");
            getEntitiesContainedIn(ents, path,_bb.getEntityFromPath(path));
            StringBuffer sb = new StringBuffer();
```

```
            if (ents!=null && ents.size()!=0){
                int i=0;
                for (EntityPrx ent : ents.values()) {
                    _logger.info(ent.getFullName(null));
                    if (i==0){
                        sb.append(ent.getFullName(null) + "," +
ent.getProperty("iName.iNameSpanish", null).getDefaultValueAsString(null) + "," +
ent.getProperty("iIcon.iconSource", null).getDefaultValueAsString(null));
                    }
                    else{
                        sb.append(";"+ent.getFullName(null) + "," +
ent.getProperty("iName.iNameSpanish", null).getDefaultValueAsString(null) + "," +
ent.getProperty("iIcon.iconSource", null).getDefaultValueAsString(null));
                    }
                    i++;
                }
            }
            else sb.append("error");

            _logger.info(sb.toString());
            return sb.toString();
        } catch (BlackBoardException ex) {
            return null;
        }

    }
```