PlanInteraction..

Arquitectura de un MAS de planificación-ejecución.

Autor: Pablo Castejón Navarro

Valencia - 11 de julio de 2012

Índice

1. PELEA.

PlanInteraction surge como propuesta de adaptación de un sistema de planificación-ejecución (PELEA), al paradigma multiagente, con el fin de estudiar el efecto de los comportamientos y las relaciones sociales en el desarrollo y ejecución de planes.

Para entender las razones que han motivado el planteamiento de la arquitectura propuesta se hace necesario introducir brevemente la arquitectura de PELEA.

PELEA es una arquitectura de planificación-ejecución que trabaja tanto a alto nivel como a bajo nivel. Esto es posible gracias a la implementación modular de la arquitectura y a una serie de traductores para la comunicación entre dichos módulos. En nuestro caso, vamos a centrarnos en la planificación de alto nivel, por lo que, en pos de la simplicidad, identificamos cuatro módulos principales que definen el funcionamiento general de la arquitectura. Estos módulos son: Execution, Monitoring, Decision Support y High-Level Replanner.

El módulo Execution (EX) realiza las funciones de captura del dominio y del problema, sensorización del estado y actuación.

El módulo Monitoring (MO) se encarga de comprobar que la ejecución de las acciones contempladas en un plan se realizan correctamente obteniendo el estado esperado.

El módulo Decision Support (DS) implementa decisiones de actuación frente a replanificaciones, correcciones o heurísticas de planificación.

Y por último, el módulo del High-Level Replanner (HLR) genera planes para un dominio y problema dados con el planificador incorporado.

A continuación se presenta el hilo típico de ejecución del sistema:

1. El módulo EX obtendría el dominio y problema y se lo comunicaría al MO, éste a su vez lo comunicaría al DS y, por último, al HLR para la obtención de un plan.
2. El HLR proporciona el plan generado al DS que aplicará las heurísticas oportunas y facilitará el nuevo plan junto las variables a monitorizar al MO. Éste último suministrará las acciones del plan al EX y realizará la acción de monitorización al término de cada acción.
3. Si el resultado de una acción no es el esperado, el MO solicitará una reparación o una replanificación por parte del DS. En caso de que todo proceda como es de esperar, la ejecución continúa con normalidad.

2. MAGENTIX 2.

Magentix2 es una plataforma multiagente que proporciona, entre otras cosas, todos los servicios de comunicación necesarios para la interacción entre agentes facilitando así el desarrollo de sistemas multiagente centrándose únicamente en el diseño de los propios agentes.

Esta plataforma provee soporte a 3 niveles: Organización, Interacción y Agente. En el nivel de Organización están contempladas las relaciones de asociación entre agentes, normas, jerarquías y sociedades. El nivel de Interacción comprende los actos comunicativos entre agentes, protocolos y ontologías de comunicación. Y por último a nivel de Agente para el desarrollo de los métodos de razonamiento y aprendizaje de los propios agentes.

La plataforma además de toda la infraestructura de comunicación proporciona otros servicios como son un sistema de trazas, seguridad, agentes conversacionales, Framework Thomas, Jason & J-Moise y herramientas de desarrollo.

La infraestructura de comunicación utiliza Apache Qpid como broker de comunicación y mensajes de comunicación según el estándar FIPA ACL. Así mismo se proveen de los mecanismos necesarios para permitir comunicación vía web mediante el protocolo http y otras arquitecturas de agentes como puede ser JADE.

Para ejecutar Magentix2, tiene que estar en funcionamiento Qpid. Los parámetros de conexión que debe proporcionar un agente son:

- <QpidHost> máquina donde se ejecuta Qpid.
- <QpidPort> puerto en que Qpid está escuchando. (por defecto 5672)
- <QpidVhost> path que actúa como namespace.
- <QpdidUser> nombre de usuario de acceso a Qpid.
- <QpidPassword> password de acceso a Qpid.
- <QpidSSL> indica si se usa SSL durante la conexión.

Los valores por defecto para estos parámetros se pueden modificar en el fichero Settings.xml.

Existen 3 métodos para realizar la conexión de los agentes: con los parámetros por defecto, definiendo únicamente el host donde se está ejecutando Qpid o definiendo el valor de todos los parámetros.

```
AgentsConnection.connect();
AgentsConnection.connect("gtiiaprojects2.dsic.upv.es");
AgentsConnection.connect(localhost, 5672, "test", "guest", "guest", false);
```

Un detalle a tener en cuenta a la hora de realizar la conexión de un agente es que no puede haber agentes con el mismo identificador, es decir, los identificadores de los agentes tienen que ser únicos.

Se proporcionan una serie de plantillas para facilitar el desarrollo de agentes:

- BaseAgent .- agente básico.
- SingleAgent .- inplementa reacción ante un mensaje recibido.
- QueueAgent .- implementa protocolo de comunicación.
- CAgent .- implementa actos comunicativos más complejos.
- BridgeAgentInOut .- agente auxiliar para comunicación al exterior.
- BridgeAgentOutIn .- agente auxiliar para comunicación desde el exterior.

2.1. Ejemplo de agentes MAGENTIX 2.

```
public class SenderAgent extends BaseAgent {
      public SenderAgent(AgentID aid)throws Exception { super(aid); }
      public void execute(){
            System.out.println("Hi! I'm agent "+this.getName());
            ACLMessage  msg = new ACLMessage(ACLMessage.INFORM);
            msg.getSender(this.getAid());
            msg.addReceiver(new AgentId("Consumer"));
            msg.setContent("Hi! I'm Sender agent and I'm running on Magentix2");
            this.send(msg);
      }
}

public class ConsumerAgent extends SingleAgent {
public ConsumerAgent(AgentID aid) throws Exception { super(aid); }
      public void execute(){
            System.out.println("Hi! I'm agent "+this.getName());
            ACLMessage msg = null;
            try{
                  msg = this.receiveACLMessage();
            }catch(InterruptedException e) { e.printStackTrace(); }
            System.out.println("Hi! I'm agent "+this.getName()+"and I've
                              received the message: "+ msg.getContent());
     }
}

public class Main {
public static void main(String[] args) {
    DOMConfigurator.configure("configuration/loggin.xml");
    Logger logger = Logger.getLogger(Main.class);
    AgentsConnection.connect("localhost", 5672, "test", "guest", "guest", false);
    try{
          SenderAgent senderAgent = new SenderAgent(new AgentID("Sender"));
          ConsumerAgent consumerAgent = new ConsumerAgent(new AgentID("Consumer"));
          consumerAgent.start();
          senderAgent.start();
    }catch(Exception e) { logger.error("Error "+e.getMessage()); }
}
}
```

3. Background.

Los sistemas multiagente son entornos **heterogéneos** (cada agente presente en el sistema puede tener un origen, métodos de razonamiento y de comunicación distintos), **dinámicos** (los agentes pueden registrarse en el sistema, abandonarlo o modificar un cierto estado mediante acciones en cualquier momento) y **comunicativos** (los agentes intercambian información para alcanzar sus metas).

Nuestro propósito es el de adaptar la arquitectura PELEA a una arquitectura multiagente y por tanto tendremos que considerar estas y otras características para hacer un buen diseño, funcional y eficiente.

PELEA es una arquitectura de planificación-ejecución pensada tanto para simulación como para ejecución real. En el diseño del nuevo modelo estará contemplada esta funcionalidad, de modo que se pueda realizar una **simulación total, simulación parcial** (agentes reales y simulados simultáneamente) y/o **ejecución real**.

Un agente de planificación-ejecución ha de tener capacidades de obtención de información, de estado del mundo, de actuación, comunicación y razonamiento. Como es habitual en el mundo de la computación y automática, la información del estado del mundo se obtiene mediante **sensores** y la modificación de un estado se realiza mediante **actuadores**. Los sensores y actuadores en un dispositivo físico están en contacto con la realidad tangible, con el **entorno real**, mientras que un dispositivo ficticio, simulado, no tiene la posibilidad de interactuar con el mundo real, por lo que se precisa de un **entorno simulado** para mantener la información de estado y poder realizar las acciones de sensorización y actuación oportunas sobre este.

Una consecuencia de la heterogeneidad y dinamismo de los sistemas multiagente es que cada agente puede tener un concepto diferente respecto del tiempo. Un sistema de planificación, sobretodo en caso de planificación temporal, tendría muchos problemas en caso de que cada agente funcionase con un reloj diferente. Además, en caso de realizar una simulación total, interesaría poder tener un reloj más rápido que en caso de una ejecución real. Es preciso establecer un **reloj** que unifique a todos los agentes en el mismo marco temporal y con la misma frecuencia.

Cada agente realizará sus funciones de planificación y ejecución sobre el entorno que le corresponda (real/simulado) y se podrá comunicar con otros agentes para su coordinación, sin embargo, necesitamos un último elemento que controle la ejecución sobre un determinado dominio/problema, que inicie la simulación/ejecución, establezca el tipo de reloj que se va a emplear y que registre a los agentes que vayan apareciendo, indicándoles aquellos agentes con los que pueden comunicarse.

3.1. Aproximación simple centralizada.

La aproximación más inmediata consistiría en encapsular todos los módulos de PELEA en un único agente que realice todo el proceso de planificación-ejecución de manera interna.

Esta aproximación no es adecuada por varios motivos:

- Complejidad de codificación: Puede haber actos comunicativos en varios puntos del proceso de planificación (DS-HLR, HLR-DS, DS-MO...) y no siempre con las mismas intenciones.
- Configurabilidad: Cada agente puede tener un método de razonamiento diferente, un planificador diferente, una forma de negociar diferente, etc. Estos métodos pueden ser muy sofisticados y no tiene sentido tener que replicar grandes cantidades de código entre agentes que solo varían algún parámetro.
- Secuencialidad: Se pierde en cierta medida el paralelismo de algunos procesos.

3.2. Aproximación simple distribuida.

En este caso lo que se plantea es todo lo contrario, aprovecharse de la modularidad de PELEA para encapsular como un agente cada uno de los módulos. Esta aproximación tiene ciertas ventajas frente a la anterior puesto que es más fácil diseñar cada agente, se pueden reutilizar agentes que ofrecen el mismo tipo de servicio, hay una mayor flexibilidad.

Sin embargo, también se ve incrementado el flujo de mensajes entre agentes, y hay que mantener de forma adecuada las comunicaciones entre agentes.

Al disponer de cada módulo de PELEA como un agente independiente se pueden realizar todo tipo de asociaciones, como un MO que controle varios EX, varios MO que utilicen un mismo DS, varios DS que requieran un mismo HLR u otras combinaciones menos habituales como varios MO que controlen un mismo EX.

3.3. Aproximación mixta.

En el conjunto de módulos que componen la arquitectura PELEA podemos distinguir unos más orientados a la ejecución (EX y MO) y otros a la planificación (DS y HLR). Una aproximación intermedia a las anteriores consistiría en plantear 2 tipos generales de agentes, los **agentes de planificación** y los **agentes de ejecución**. Los agentes de planificación comprenderían los módulos DS y HLR, mientras que los agentes de ejecución tendrían los módulos EX y MO.

Mediante esta agrupación se consigue reducir el flujo de mensajes en el sistema, sacrificando cierta flexibilidad en cuanto al tipo de configuraciones posibles. En este tipo de aproximación se considera un único MO para comprobar el resultado de la ejecución de las acciones de un único agente, sin embargo es posible que varios agentes de ejecución se comuniquen con un único agente de planificación para realizar un plan coordinado o que cada uno se comunique independientemente con un agente de planificación diferente, mostrando un comportamiento más egoísta o independiente.

3.4. Visibilidad.

Habitualmente los agentes de un sistema multiagente cuentan con un conocimiento parcial de la información, en nuestro caso del estado del mundo. Sin embargo, el conjunto de toda la información distribuída entre el conocimiento de los agentes presentes en el sistema puede no ser completa, puede existir información susceptible de ser descubierta por otros medios que no sea mediante comunicación entre agentes. Para ello se precisa de ciertas capacidades de adquisición de información que en el mundo de la automática proveerían los sensores.

Esta capacidad de percepción del entorno o de captación de información se puede asociar con la visibilidad de un agente. Podemos diferenciar 3 tipos de visibilidad:

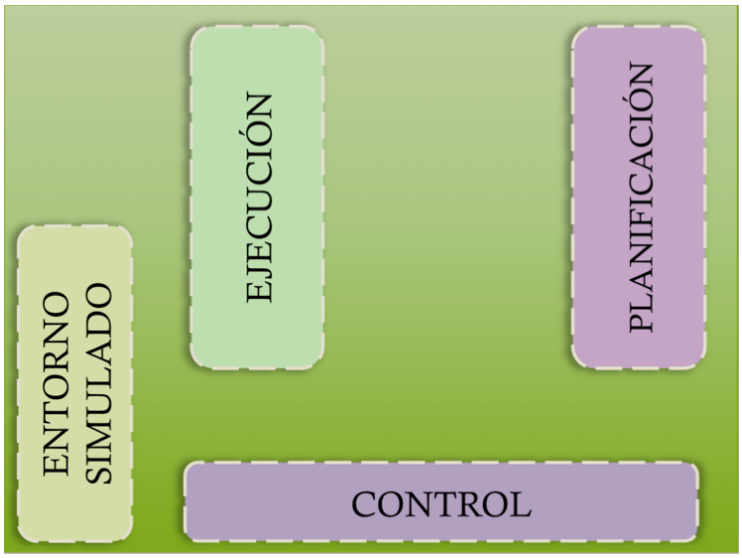
- **Extensión**. Esta visibilidad hace referencia al conjunto de valores que un agente conoce del dominio de posibles valores para una determinada variable. Puede ser total, parcial o nula, según si conoce todos los posibles valores del dominio, algunos o ninguno respectivamente.
- **Intercambio**. La información que conoce un agente y que está dispuesto a compartir con otros agentes mediante comunicación, ya sea para negociar o coordinarse.
- **Adquisición**. Las capacidades que tiene el agente para obtener información del entorno definen este tipo de visibilidad. Los agentes físicos tendrán sus sensores reales, mientras que los agentes simulados precisan de una definición de estas capacidades, por ejemplo mediante patrones condicionales.

4. Arquitectura (v.1.0-29/06/2012).

La arquitectura va a permitir la implementación de agentes al modo de la aproximación simple distribuida, aportando las clases necesarias para la definición de cada tipo de agente. Sin embargo, en nuestra implementación vamos a trabajar con la aproximación mixta en la que se dispone de agentes de planificación y agentes de ejecución.

4.1. Modelo.

Se plantea un modelo de arquitectura compuesto por 4 módulos: control, simulación, ejecución y planificación.



Los módulos de planificación y ejecución están motivados por la aproximación escogida para la adaptación de PELEA a un MAS. En el módulo de planificación se agruparían los agentes de planificación incluyendo un DS y un HLR, mientras que en el módulo de ejecución situaríamos los agentes de ejecución compuestos por un MO y un EX.

El módulo de control contendrá todo aquello necesario para el correcto funcionamiento y configuración del sistema.

Por último, el módulo de simulación nos permitirá mantener una representación del entorno en el que se ejecutan las acciones de los agentes.

4.1.1. Control.

El módulo de control se compone principalmente de dos elementos, el **Session Controller** (SC) y un agente con funciones de reloj.

El SC es el encargado de establecer las configuraciones iniciales del sistema, de mantener la información de los dominios de planificación en ejecución y los agentes presentes en el mismo.

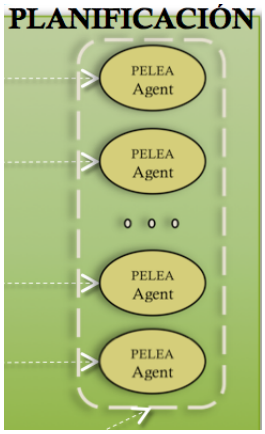
Al inicio del sistema el SC será el encargado de establecer qué tipo de reloj se manejará en el MAS: si será un reloj continuo o fraccionado, la velocidad del mismo, si se permiten interrupciones o no, etc. Estos detalles son importantes a tener en cuenta porque restringirán el tipo de agentes, interacciones y dominios de planificación que se podrán tratar en el MAS.

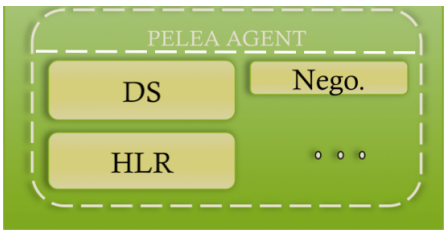
En el modelo más básico el reloj, representado como un agente, tiene las funciones de sincronizar a los agentes presentes en el MAS, tanto en el momento de su registro como ante cualquier cambio provocado durante la ejecución. El sincronizar a los agentes en su entrada al MAS permite que cada agente mantenga internamente su concepción del tiempo y no se tengan que estar intercambiando mensajes continuamente con el reloj del MAS, sin embargo, ante una inconsistencia entre el reloj interno de un agente y el del MAS, prevalecerá el de este último.

Otra de las funcionalidades que tendrá el SC será el de registrar a los agentes que van dándose de alta en el MAS de tal forma que se les facilite con qué otros agentes puede comunicarse, ya sea por cuestión de dominio de problema o por preferencias de la tipología de los agentes y/o de planificación.

4.1.2. Planificación.

El módulo de planificación comprende los agentes de planificación y sus comunicaciones. Partiendo de la división modular de PELEA, los agentes de planificación que definimos en nuestro modelo han de contener las capacidades del DS y del HLR. A estos agentes los llamaremos agentes PELEA.





Los agentes PELEA, además de cubrir las funcionalidades propias del DS y del HLR de PELEA, tendrán que tener definidas otras funcionalidades propias de los agentes, como métodos de razonamiento, comunicación, negociación y otras técnicas sociales.

Un agente PELEA, por sí mismo no tiene ningún objetivo inicial, pero sí que tendrá un comportamiento y un tipo de planificador propios. Los objetivos de un agente PELEA vendrán definidos por el problema a resolver que le facilite uno o más agentes de ejecución. Cuando un agente PELEA recibe un problema ha de resolverlo según sus propios métodos de razonamiento para ofrecer al agente de ejecución la que considere mejor solución.

La principal tarea de un agente PELEA se puede entender como ofrecer a un agente de ejecución un plan para cumplir sus objetivos. En la elaboración de este plan pueden intervenir muchos factores que hagan que se elabore un determinado plan u otro. Uno de los factores más obvios es el tipo de planificador asociado al agente (lpg, ff, sgplan...). Pero la parte más importante que diferencia socialmente a un agente PELEA de otro será el tipo de coordinación y/o negociación que implemente.

Un agente PELEA puede tener que generar un plan para un único agente de ejecución (1), generar un plan coordinado para varios agentes de ejecución (2) o bien coordinarse con otros agentes PELEA para elaborar planes coordinados (3).

El primero de los casos es el más sencillo e intuitivo, se recibe un problema, se genera un plan y se devuelve.

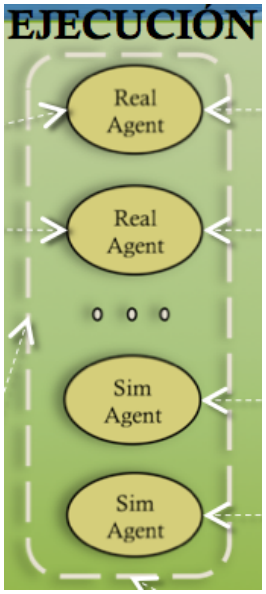
El segundo de los casos supone que varios agentes de ejecución solicitan al mismo agente PELEA la elaboración de un plan coordinado para resolver sus objetivos. Esto puede venir motivado porque varios agentes de ejecución determinen utilizar el mismo planificador como resultado de alguna coordinación previa o bien como extensión del primer caso, en que un agente de planificación va recibiendo en distintos instantes de tiempo nuevos objetivos provenientes de diferentes agentes de ejecución. Como resultado de utilizar un mismo planificador para varios agentes de ejecución, se obtiene un plan coordinado que tendrá que ser repartido entre los distintos agentes de ejecución que participan del mismo.

El último caso es el más complicado, puesto que los agentes PELEA deben determinar con qué agentes pueden colaborar para cumplir sus objetivos y comunicarse siguiendo un protocolo y métodos de negociación preestablecidos. En este caso habrá que definir el tipo de coordinación que se llevará a cabo: antes, durante o tras la generación del plan. La coordinación del plan previa generación del mismo pasa por hacer un reparto de los objetivos entre los agentes. La coordinación durante la generación del plan consiste en construir de forma conjunta el plan resolviendo cada agente los subobjetivos que van apareciendo en la consecución de los objetivos globales. Y la coordinación tras la generación de planes consiste en la combinación de los planes de tal forma que no haya inconsistencias entre ellos (plan-merge).

En resumen, un agente de planificación deberá contar con un planificador determinado, una estrategia de negociación y un método de coordinación de la planificación estático o dinámico.

4.1.3. Ejecución.

En el módulo de ejecución agrupamos a los agentes con capacidades de ejecución, en relación con PELEA los agentes de ejecución tendrán, al menos, las funcionalidades del MO y del EX.



Los agentes de ejecución son los principales representantes del MAS, son los que tienen la especificación inicial del dominio y del problema que quieren resolver, solicitan planes a los agentes de planificación para poder ejecutarlos y lograr así sus objetivos.

Por tratarse de agentes de ejecución, cuentan con unas capacidades de acción o realización de funciones que modifican el estado de algo. Este algo que modifica su estado puede ser físico/tangible o simulado, por ello diferenciamos entre **agentes reales** y **agentes simulados**.

Los agentes reales pretenden ser representantes en el MAS de entidades físicas, como pueden ser robots, que deben realizar una determinada tarea en el mundo real. De momento no entraremos en mucho detalle sobre este tipo de agentes, simplemente decir que perciben y actúan sobre el mundo real y deben ser consistentes con los agentes simulados en caso de que los hubiera.

Los agentes simulados, por otro lado, no son capaces de percibir ni actuar sobre el mundo real, no cuentan con sensores ni actuadores físicos. Sin embargo, para la correcta simulación de estos agentes, deben ser capaces de capturar el estado del mundo y realizar modificaciones sobre el mismo de alguna manera. Por ello, el MAS dispone de una representación simulada del estado del mundo que se comentará más adelante.

Durante la ejecución de un plan, un agente de ejecución debe monitorizar la ejecución para comprobar que todo resulta como es de esperar. En caso con-

trario, implementará los métodos oportunos para solucionar el problema causado, bien a nivel de ejecución o bien solicitando reparación o replanificación a algún agente de planificación.

Cuando un agente de ejecución quiere acceder al MAS debe registrarse en el SC, indicando el dominio del problema que va a tratar. De esta forma se le puede proporcionar un listado de los agentes que comparten su dominio de planificación y facilitar así las tareas de comunicación para la coordinación. Así mismo, durante el registro, el agente reloj le suministrará el tiempo del sistema y sus características para que mantenga su reloj interno actalizado y coherente con el del MAS.

Para poder facilitar el dominio de planificación al SC durante el proceso de registro, un agente de ejecución debe obtenerlos de un fichero codificado en PDDL. La versión de PDDL con la que trabajaremos será una versión de PDDL 3.1 extendida para permitir la definición de capacidades extras dependientes del agente. Estas extensiones codificarán los criterios de visibilidad de intercambio y de adquisición del agente. Las capacidades de adquisición del agente serán comunicadas al entorno simulado para simplificar el proceso de obtención de información del mismo.

Otro de los criterios que debe definir un agente de ejecución es el tipo de agente de planificación con el que quiere relacionarse, pudiendo así el SC facilitarle aquellos que cumplen las características y seleccionar uno.

4.1.4. Simulación.

En el módulo de simulación tenemos la representación del estado del mundo. A esta representación la llamamos entorno simulado. El entorno simulado permite a los agentes simulados obtener el estado de los distintos elementos del problema, así como modificarlo mediante la ejecución de las acciones de sus planes.

Un entorno simulado ha de mantener la información de estado de todos los elementos de un problema, incluso de aquellos elementos que ningún agente conozca a priori pero sean susceptibles de llegar a conocerse. Esta información se le suministra en el momento de su creación. Ante una petición de un agente de ejecución de sensorización del entorno, el entorno simulado devolverá únicamente la información pertinente atendiendo a las capacidades de sensorización del agente en cuestión.

La información relativa a las capacidades de sensorización de un agente de ejecución serán comunicadas al entorno simulado en el proceso de registro del agente. De este modo se simplifican las comunicaciones entre agentes de ejecución y entorno simulado haciendo una petición genérica de estado del mundo. El entorno simulado utilizará las capacidades definidas de cada agente para filtrar la información que le debe suministrar ante una petición de estado.

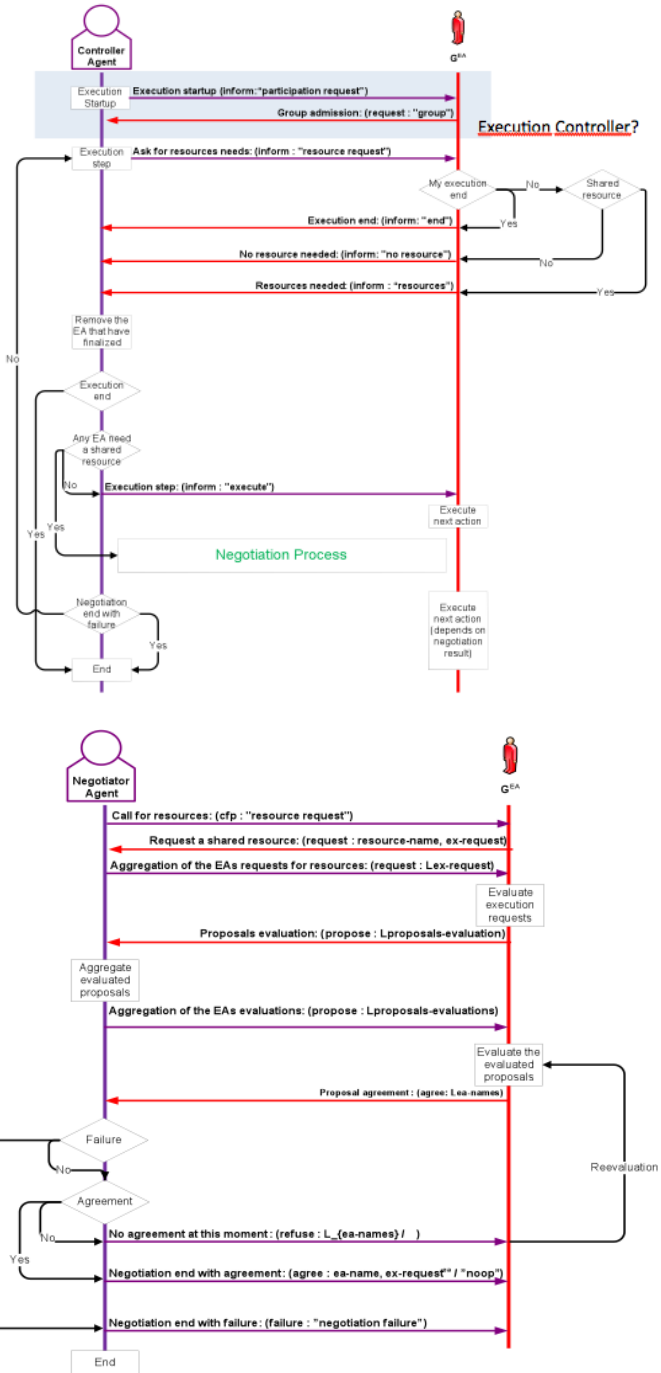
Para poder permitir la ejecución simultánea de varios dominios de planificación, se hace necesario la implementación de un entorno de planificación específico del dominio. Si no fuera así tendríamos que lidiar con problemas de gestión de nombres de variables, para no permitir tener dos elementos con la misma referencia y que agentes que pertenecen a un determinado dominio obtengan información de otro dominio.

Cada agente de ejecución podrá consultar de entre los entornos simulados presentes en el MAS, cuál ofrece soporte a su dominio concreto y comunicarse con él.

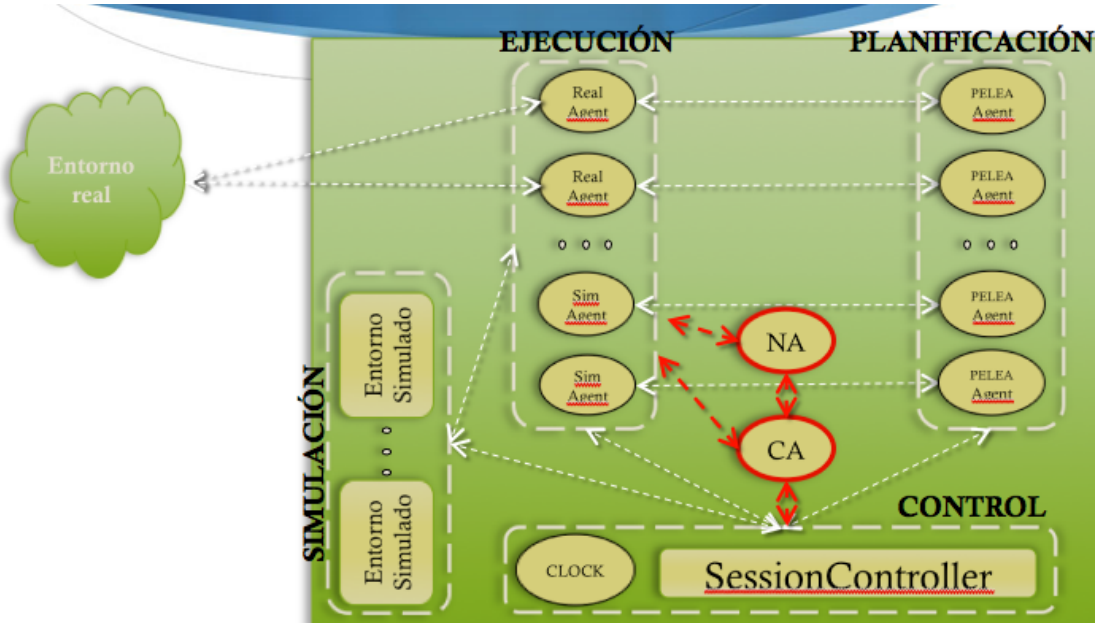
4.1.5. Negociación.

Adicionalmente se propone un servicio de negociación previa ejecución orientada al uso de recursos, es decir, los agentes de ejecución que vayan a utilizar algún recurso del problema en las sucesivas acciones de su plan podrán utilizar dicho servicio de negociación para coordinarse con otros agentes.

Este servicio consta de un Controller Agent (CA) y uno o más Negotiator Agent (NA). El CA hace de manager de negociaciones controlando en cada instante temporal qué agentes de ejecución deben negociar la utilización de algún recurso. Para ello los agentes de ejecución deberán comunicar previamente al CA la lista de recursos que pueden requerir y el plan de acciones que van a realizar. En caso de que varios agentes de ejecución vayan a requerir el mismo recurso en el mismo tiempo de ejecución, el CA crea un NA para gestionar la negociación sobre el uso del recurso entre ambos agentes. Al finalizar todas las posibles negociaciones para un instante temporal, el CA comunica a los agentes de ejecución que pueden proceder con la ejecución de la siguiente acción.


Controller Agent
G^EA
Execution Startup
Execution startup (inform:"participation request")
Group admission: (request : "group")
Execution Controller?
Execution step
Ask for resources needs: (inform : "resource request")
My execution end
No
Shared resource
Yes
Execution end: (inform: "end")
No resource needed: (inform: "no resource")
No
Resources needed: (inform : "resources")
Yes
Remove the EA that have finalized
No
Execution end
Any EA need a shared resource
No
Execution step: (inform : "execute")
Execute next action
Yes
Yes
Negotiation Process
Negotiation end with failure
Yes
End
Execute next action (depends on negotiation result)
Negotiator Agent
G^EA
Call for resources: (cfp : "resource request")
Request a shared resource: (request : resource-name, ex-request)
Aggregation of the EAs requests for resources: (request : Lex-request)
Evaluate execution requests
Proposals evaluation: (propose : Lproposals-evaluation)
Aggregate evaluated proposals
Aggregation of the EAs evaluations: (propose : Lproposals-evaluations)
Evaluate the evaluated proposals
Proposal agreement : (agree: Lea-names)
Reevaluation
Failure
No
Agreement
No
No agreement at this moment : (refuse : L_{ea-names} /)
Yes
Yes
Negotiation end with agreement: (agree : ea-name, ex-request" / "noop")
Negotiation end with failure: (failure : "negotiation failure")
End

Viendo el modelo de arquitectura en conjunto tendríamos algo como lo que se muestra en la siguiente imagen.



4.2. Visibilidad.

Ya se ha definido previamente en la sección 3.4 el concepto de visibilidad y sus tipos, por lo que en esta sección nos vamos a centrar más en detallar las estructuras que necesitamos para gestionar este tipo de información y particularizar cada tipo de visibilidad al modelo que planteamos.

En la sección 4.1.4 se ha indicado que los agentes de ejecución adquieren la información del dominio y del problema en formato PDDL 3.1 extendido. Esto está motivado por la gestión de la visibilidad. La versión 3.1 de PDDL consiste fundamentalmente en una extensión de los tipos de datos fluents para manejar objetos en lugar de enteros. De esta forma se permite hacer una representación del conocimiento con variables de estado. Además, las variables pueden no estar inicializadas, permitiendo representar así la incertidumbre o desconocimiento de cierta información. Esto nos permite a su vez hacer uso de la lógica triestado, según la cual una información puede ser cierta, falsa o desconocida.

En el entorno simulado tan solo estará representado aquello que es cierto, dejando la información falsa o desconocida para la representación interna de los agentes de ejecución.

La información que recibe un agente de ejecución como respuesta a una acción de sensorización es siempre cierta. Esta información cambiará el estado interno del agente en función del conocimiento previo que tuviera. Si un agente obtiene que el valor de una variable A es V_1, el agente automáticamente sabe que el resto de valores del dominio de A son falsos. Si por el contrario no recibe información del valor asociado en un determinado instante de tiempo a A, se crea una situación de incertidumbre, puesto que pese al conocimiento previo que este agente tuviera sobre el valor de A no es capaz de afirmar que mantenga el mismo valor. La información fruto de estas situaciones de incertidumbre puede ser tratada como información desconocida o información sujeta a instantes temporales, de cara a la interacción con otros agentes.

La visibilidad de un agente, sea del tipo que sea, se verá reflejado en el fichero del problema. Esto es así para permitir la definición de agentes con el mismo dominio de planificación, pero con diferentes capacidades de visibilidad.

4.2.1. Extensión.

Este tipo de visibilidad es una capacidad pasiva, un agente no puede determinar el tipo de visibilidad que tiene respecto de una variable o modificarla, puesto que es una medida del conocimiento del agente dentro del conocimiento universal del dominio del problema. Sin embargo, a modo de especificación y formalización se pueden distinguir 3 tipos de visibilidad de extensión: total, parcial y nula.

Asumiendo una variable V con dominio de valores $D_V = \{d_1, d_2, \ldots, d_n\}$, un agente A tendrá **visibilidad total** respecto de V si su dominio de valores conocidos para V es $D_{A,V} = D_V$.

Asumiendo una variable V con dominio de valores $D_V = \{d_1, d_2, \ldots, d_n\}$, un agente A tendrá **visibilidad parcial** respecto de V si su dominio de valores conocidos para V es $D_{A,V} \subset D_V \wedge D_{A,V} \neq \emptyset$.

Asumiendo una variable V con dominio de valores $D_V = \{d_1, d_2, \ldots, d_n\}$, un agente A tendrá **visibilidad nula** respecto de V si su dominio de valores conocidos para V es $D_{A,V} = \emptyset$.

4.2.2. Intercambio.

La visibilidad de intercambio permite restringir el conocimiento que un agente está dispuesto a compartir con otros agentes. PDDL no permite representar este tipo de información directamente, por lo que definimos una extensión a tal efecto.

Introducimos la sección **shared-data:** del fichero de problema, en ella se detallarán mediante patrones aquella información que un agente está dispuesto a compartir y con quién. Si no se indica explícitamente con qué agentes se comparte la información que casa con un determinado patrón, esta información se comparte con todos los agentes. A continuación se muestra un ejemplo de la sintaxis que siguen los patrones incluidos en esta sección.

```
(:shared-data
    (hung ?p - picture)
    ((posA ?a - agent) - location)
    ((posP ?p - picture) - location)
    ((posT ?t - tool) - (either agent location)) - ag2)
```

Este tipo de visibilidad define cierto comportamiento social de un agente, ya sea un agente de ejecución o un agente de planificación. Asumimos que todos los agentes comparten información cierta según su conocimiento, aunque también existe la posibilidad de ocultar información, es decir, no querer compartir cierto conocimiento.

Respecto de la veracidad de la información compartida por los agentes mediante los patrones de visibilidad, hay que destacar que el estado del mundo está en continuo cambio con el transcurso del tiempo y los agentes no están monitorizando de forma continua el estado, esto sería muy costoso. Por tanto, el conocimiento que poseen los agentes y que pueden compartir es cierto en relación a un determinado instante de tiempo pasado. Es de suponer que a información contradictoria con diferentes fuentes de procedencia sobre un mismo elemento tendrá mayor credibilidad aquella que tenga una referencia temporal posterior.

4.2.3. Adquisición.

La visibilidad de adquisición pretende modelar las capacidades sensitivas que tienen los agentes, es decir, la información que puede adquirir un agente del entorno mediante sensorización. Existen muy diversos tipos de sensores que permiten la obtención de un tipo de información u otra (auditiva, visual, electromagnética, etc.). En el caso de los agentes simulados no se dispone de los sensores físicos, por lo que se precisa modelar de alguna forma las capacidades que estos otorgan. Es tarea del diseñador de un agente de ejecución definir adecuadamente las capacidades de sensorización de que dispone.

Para llevar a cabo la definición de estas capacidades de sensorización, extendemos nuevamente el PDDL con otra serie de secciones, una por sensor, que se indicarán con la etiqueta **:sensing** seguida del nombre del sensor concreto que se esté definiendo. Esta independencia en la definición de los sensores nos permitirá hacer consultas más especializadas en caso de querer únicamente la información que aportan un determinado tipo de sensor.

Las secciones *:sensing* contendrán patrones, al igual que en la extensión *shared-data:*, pero con la particularidad de ser condicionales. Es preciso establecer una serie de condiciones puesto que no toda la información del entorno simulado que case con un patrón definido es accesible para un agente de ejecución en todo momento, por ejemplo, una cámara tradicional podría aportar conocimiento sobre lo que ve inmediatamente delante suya, pero no de lo que se encuentra tras una pared, a pesar de que haya podido estar previamente tras ella.

De este modo un agente de ejecución simulado, cuando quiera obtener información del entorno simulado correspondiente realizará una acción de sensorización de los sensores pertinentes, enviando al entorno simulado los patrones de dichos sensores para que actúen como filtro de la información que éste debe devolver.

A continuación se muestra un ejemplo de patrones definidos para dos camaras.

```
(:sensing camera1
    (at ?p - pkt) - (= (at oscar) (at ?p)))

(:sensing camera2
    (at ?p - pkt) - (and (= (at oscar) ?l - location)
                         (= (at ?p) ?l )))
```

Los patrones anteriores permiten obtener aquellos paquetes que se encuentran en la misma localización que *oscar*.