Una Extensión de Agentes en JASON para Razonar con Incertidumbre: G-JASON

Adrián Biga[1] and Ana Casali[1,2]

[1] Facultad de Cs. Exactas, Ingeniería y Agrimensura
Universidad Nacional de Rosario (UNR)
Av. Pellegrini 250 - S2000BTP, Rosario, Argentina
[2] Centro Internacional Franco Argentino de
Ciencias de la Información y de Sistemas (CIFASIS)
aebiga@gmail.com, acasali@fceia.unr.edu.ar

Abstract. Una de las mejores implementaciones de agentes de la familia BDI (B: Belief, D: Desire, I: Intention) es mediante los llamados Sistemas de Razonamiento Procedural (PRS). En este trabajo se plantea una extensión de los PRS para permitir generar agentes más flexibles, que puedan representar la incertidumbre del entorno y distintos grados de relevancia en los planes del agente. La extensión propuesta se implementó en la plataforma JASON, que permite la implementación de agentes PRS en JAVA otorgándoles alta portabilidad.

Keywords: Agentes de Razonamiento Procedural, JASON, modelo BDI, incertidumbre

1 Introducción

En los últimos años ha crecido el interés en modelar sistemas complejos como sistemas multiagentes [10]. Dentro de las arquitecturas más notorias para dar soporte a los agentes que componen estos sistemas, se encuentra la arquitectura BDI (B: Belief, D: Desiere, I: Intention)([9],[7]). Esta arquitectura de agentes ha sido una de las más estudiada y utilizada en distintas aplicaciones reales de importancia [4].

Los Sistemas de Razonamiento Procedural (PRS) [6], son las implementaciones más conocidas de una arquitectura basada en el paradigma BDI. Desde la primera implementación de un sistema PRS [5], se han desarrollado distintas versiones, respecto a implementaciones en Java destacamos dos versiones actualmente utilizadas: JACK [3] y JASON [1].

Una de las limitaciones tanto el modelo BDI definido por Rao y Georgeff [9] como los sistemas basados en PRS [5], es que no contemplan una forma explícita de representar la incertidumbre del entorno, estos se han planteado considerando una lógica bi-valuada para dar soporte a toda la información que utiliza un agente para la toma de decisiones (representada en sus estados mentales: B, D e I).

[3] http://aosgrp.com/products/jack/

En el modelo BDI graduado (g-BDI) presentado en Casali et al.[3, 2], se plantea un modelo general de agente BDI especificando una arquitectura que pueda tratar con la incertidumbre del entorno y actitudes mentales graduadas. De esta forma los grados de las creencias van a representar en que medida el agente cree que una fórmula es cierta, los grados de los deseos permiten representar el nivel de preferencia y el grado de las intenciones darán una medida costo-beneficio que le representa al agente alcanzar cada objetivo. Para especificar la arquitectura de un agente BDI graduado, se utiliza la noción de sistema multicontextos. De esta forma, un agente g-BDI es definido como un grupo de contextos interconectados para representar sus creencias, deseos e intenciones. Cada contexto tiene su lógica asociada, para representar y razonar con grados en las creencias, deseos e intenciones, se ha elegido utilizar lógicas modales multivaluadas. Las características generales de los distintos componentes de un agente BDI graduado se pueden ver en [3]. Se ha observado que el modelo g-BDI brinda un framework que permite modelizar agentes más flexibles y que pueden representar apropiadamente la incertidumbre del entorno y preferencias del agente [4], pero aún no se ha desarrollado una plataforma que permita una implementación genérica de los agentes especificados bajo este modelo.

Con inspiración en este modelo de agentes g-BDI, en este trabajo se presenta una extensión de los sistemas PRS para representar estructuras de datos graduadas en sus creencias (Beliefs) y en los planes del agente.

2 Arquitecturas de Agentes y Sistemas PRS

Existen diferentes propuestas para la clasificación de arquitecturas de agentes. Tomando como referencia la clasificación realizada por Wooldrige [10], se puede considerar las siguientes clases concretas de arquitecturas: deliberativas, reactivas, híbridas y arquitecturas de razonamiento práctico. Dentro de este último grupo se encuentran los agentes BDI.

La arquitectura BDI está caracterizada porque los agentes están dotados de estructura de datos que representan explícitamente los siguientes estados mentales:

- Creencias (Beliefs): representan el conocimiento que el agente tiene sobre sí mismo y sobre el entorno.
- Deseos (Desires): son los objetivos que el agente desea cumplir.
- Intenciones (Intentions): se puede considerar como un subconjunto de deseos consistentes entre sí que el agente decide alcanzar. Las intenciones derivan en las acciones que ejecutará el agente en cada momento.

La arquitectura PRS desarrollada por Georgeff y Lansky [6] fue quizás la primera arquitectura basada en el paradigma BDI. Ha sido utilizada en varias aplicaciones ([7],[8]). Un agente con arquitectura PRS trata de alcanzar cualquier meta que se proponga basándose en sus creencias sobre el mundo (entorno). También puede simultáneamente reaccionar ante la ocurrencia de algún nuevo evento. De esta forma, PRS provee un marco en el cual los comportamientos de tipo *dirigido a la meta* y *dirigido por los eventos* pueden ser fácilmente integrados. Los sistemas

PRS consisten en un conjunto de herramientas y métodos, para la representación y ejecución de planes. Estos planes o procedimientos son secuencias condicionales de acciones las cuales pueden ejecutarse para alcanzar ciertos objetivos, o reaccionar en situaciones particulares.

3 La Arquitectura de JASON

El lenguaje interpretado por JASON es una extensión del AgentSpeak(L) [1] , un lenguaje abstracto que tiene una concisa notación y es una eficiente extensión de la programación lógica para sistemas BDI. JASON a comparación de otros sistemas de agentes BDI posee la ventaja de ser multiplataforma al estar desarrollado en el lenguaje JAVA.

Un agente en **AgentSpeak (L)** es creado especificando un conjunto de creencias (beliefs) y un conjunto de planes (plans). Otros elementos relevantes son los objetivos (goals) del agente y los eventos disparadores (trigger events) que sirven para representar la parte reactiva de un agente.

- **Creencias:** representan las creencias del agente respecto a su entorno.
- **Objetivos:** representan los objetivos del agente, AgentSpeak (L) distingue sólo dos tipos de objetivos (goals): *achivement goals* y *test goals*. El *achivement goal* denota que el agente quiere alcanzar un estado en el mundo donde el predicado asociado sea verdadero. Un *test goal* devuelve una unificación asociada con un predicado en el conjunto de creencias del agente, si no hay asociación, simplemente falla.
- **Evento disparador** (*trigger event*) es un evento que puede iniciar la ejecución de un plan. Un evento puede ser interno, cuando un sub-objetivo tiene que ser logrado, o externo, cuando es generado por actualizaciones de creencias debido a una percepción del ambiente.
- **Planes:** son acciones básicas que un agente puede realizar sobre su ambiente. Un plan está formado por un evento disparador (denotando el propósito del plan), seguido por una conjunción de literales de creencia representando un contexto. Para que el plan sea aplicable, el contexto debe ser una consecuencia lógica de las actuales creencias del agente. El resto del plan es una secuencia de acciones básicas o subobjetivos que el agente tiene que lograr (o testear) cuando el plan es elegido para su ejecución.

La sintaxis de AgentSpeak(L) está definida por la gramática que se muestra en la Figura 1. Un agente *ag* es especificado como un conjunto de creencias *bs* (la base de creencia inicial) y un conjunto de planes *ps* (la librería de planes del agente). Un plan esta definido por $p ::= te : ct < -h\ p$, donde *te* es un evento disparador, *ct* es el contexto del plan y *h* es una secuencia de acciones, objetivos, o actualizaciones de creencias; *te:ct* es la cabeza del plan y *h* es el cuerpo.

La Figura 2 describe cómo trabaja un intérprete de AgentSpeak (L). En cada ciclo de interpretación de un programa de agente, se actualiza la creencia del agente y con ello la lista de eventos, que pudo ser generado por una percepción del ambiente (evento externo) o por la ejecución de intenciones(acciones), lo cual

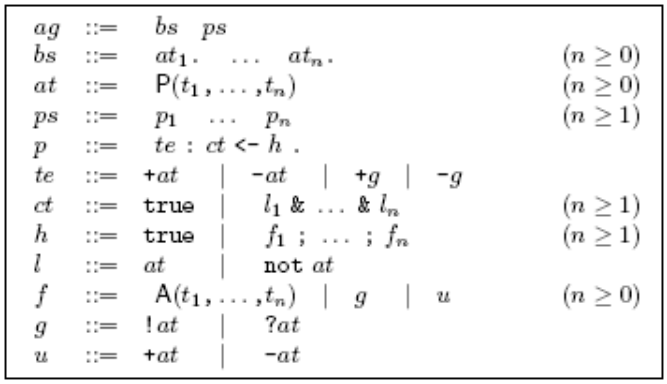

Fig. 1. Sintaxis de AgentSpeak
Fuente: [1]

produce un evento interno (ver (1) en la Figura 2). Después de que la *función Selectora de Eventos* (SE) haya seleccionado el evento (2), el agente tiene que unificar este evento con eventos disparadores en la cabeza de los planes (3). Esto genera un conjunto de planes relevantes con sus respectivos contextos, se eliminan aquellos cuyos contextos no se satisfagan con la base de creencias del agente, para formar un conjunto de planes aplicables (4). Este conjunto de planes se denominan "'opciones"'. Estas opciones se generaron a partir de un evento externo o interno. Entonces, la *función Selectora de Opciones SO* elige uno de estos planes (5), luego pone este plan con alguna intención existente (si el evento fue interno) o crea una nueva intención (si el evento fue externo). Todo lo que queda es seleccionar una intención (6) para ser ejecutada en el ciclo (7). La función Selectora de intenciones (SI) se encarga de esto. La Gramática de Jason presenta mejoras respecto a la de AgentSpeak (L) y se puede ver su definición completa en [1]. Siempre se deberá tener en cuenta que el estado de un agente, a lo largo de su ciclo de vida, está representado por un conjunto de creencias y un conjunto de planes y su comportamiento estará reflejado en el comportamiento de las funciones selectoras.

Una de las diferencias más importante que distingue a JASON de la sintaxis de AgentSpeak(L) (ver Figura 1) es la inclusión de las anotaciones tanto en las creencias como en los planes. Las anotaciones en las creencias y en los planes pueden generarse por $atomic\_formula$ mediante las siguientes reglas BNF:

$$beliefs- > (literal \text{“.”})^*; (3.1)$$
$$literal- > [\sim] atomic\_formula \ (3.2)$$
$$plan- > [@atomic\_formula] triggering\_event [: context] [\text{“} < - \text{”} body] \text{“.”} \ (3.3)$$
$$atomic\_formula- > (< ATOM > | < VAR >) [\text{“(”} list\_of\_terms \text{“)”}] [\text{“[”} list\_of\_terms \text{“]”}]; (3.4)$$

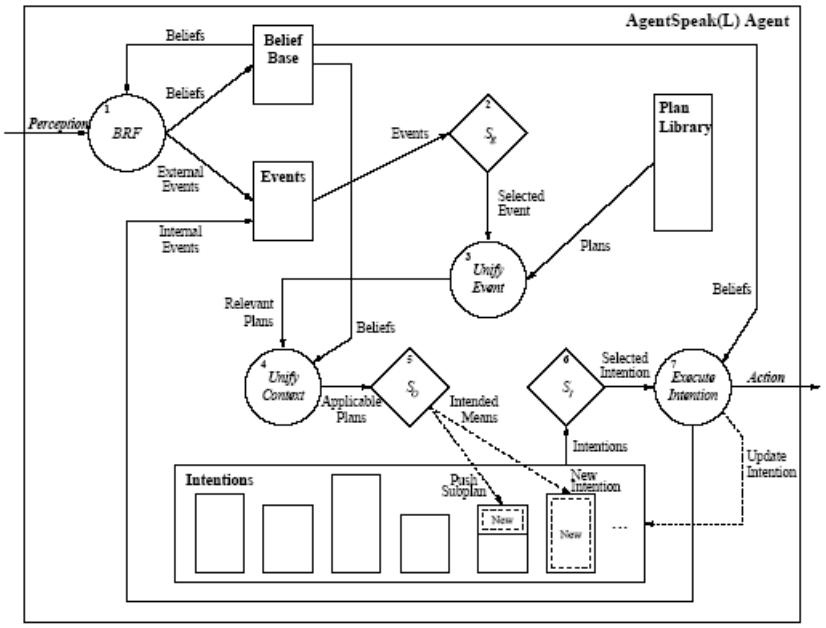

Fig. 2. Diagrama del intérprete de un agente AgentSpeak(L)
Fuente: [1]

Las fórmulas atómicas pueden contener anotaciones, esta es una lista de términos encerradas en corchetes inmediatamente siguiendo una fórmula (ver regla 3.4). Los planes tienen etiquetas representados por [@$atomic\_formula$] (ver regla 3.3).

4 Extensión G-JASON

Con la inspiración del modelo g-BDI, se presenta una extensión sintáctica y semántica de JASON que permita representar grados en las creencias y en los planes, ya que estos son los elementos fundamentales en su arquitectura:

1. *Grados en las creencias:* se agregan valores numéricos en el interválo [0,1] para representar el grado de certeza de que un hecho sea cierto, para ello se utilizan las anotaciones. Luego, una creencia queda definida por:
   $X[degOfCert(valor_x)]$ donde X representará una fórmula atómica (representando un hecho) y $valor_x$ su grado de certeza.
   Por ejemplo, la creencia de que hay un viento leve con una certeza de 0.7 se representará como: viento_leve [degOfCert(0.7)].
   Este grado impactará en el orden de selección de los eventos debido a que la función selectora SE tomará el de mayor grado primero.
2. *Grados en los Planes:* el plan está formalizado por:
   $@label[planRelevance(gradoLabel)]te[degOfCert(gradoTE)] :$
   $ct[degOfCert(gradoCT)] <- body;$

Se permite asociar a cada plan tres valores: el grado en la anotación del label (gradoLabel), en el evento disparador (gradoTE) y en el contexto (gradoCT):

- *gradoLabel* representa la medida de éxito del plan (planRelevance), este valor influye en la selección de intenciones y en la selección de opciones. De modo de que si dos planes han pasado los filtros debido a las unificaciones se ejecutará el que tenga el *gradoLabel* más alto.
- *gradoTE* es un grado aplicado al evento disparador e influye en la selección de eventos y en la unificación de eventos.
- *gradoCT* es un grado en el contexto del plan e influirá en la unificación del contexto.

Tanto el gradoTE aplicado al evento disparador, como el gradoCT aplicado al contexto, representan precondiciones adicionales: en que grado estas creencias deben ser ciertas en la base de creencias actual para que el plan sea aplicable.

Al incluir los grados mencionados, para que los mismos tengan el valor semántico deseado, se tienen que modificar las funciones selectoras descriptas y los procesos de unificación para los eventos y los contextos de los planes. A continuación se verá como se extendió la sintaxis de JASON para obtener las nuevas funcionalidades.

4.1 La Gramática de G-JASON

Las palabras "planRelevance" (relevancia de un plan) y "degOfCert" (grado de certeza) serán palabras reservadas y los grados serán representados por números flotante en el interválo [0,1]. Se modifica la sintaxis original de JASON para las creencias, planes, eventos disparadores (te) y contextos (ct):

1. Creencias: se extienden para permitir que tengan un grado de certeza:
   $beliefs ->$ (**literalC** ".") $*$ (*en la sintaxis original* *regla* 3.1)
   $literalC ->$ [" $\sim$ "]$atomic\_formula$ **anotC**
   $anotC ->$ "[$degOfCert$(" $< valor >$ ")]"
   $valor -> dig1, dig2$
   $dig1 -> 0|1$
   $dig2 -> 0|1|...|9$
2. Evento disparador (TE) del plan: se permite que sea graduado, para ello se especializa la anotación para representar el grado de certeza:
   $triggering\_event ->$ (" + " | " − ") ["!" | "?"] **literalC**
3. Contexto: se realiza el cambio de manera más directa reutilizando **anotC**
   $context ->$ log_expr **anotC** | "*true*"
4. Planes: se agrega el grado (planRelevance) dentro del plan
   $plan ->$ "@" $atomic\_formula$ **anotG** $triggering\_event$ [: $context$]
   [" $< -$"$body$] "."
   $anotG ->$ "[$planRelevance$(" $< valor >$ ")]"

4.2 Implementación de las extensiones: G-JASON

Se realizaron las modificaciones en el código de JASON tanto sobre los beliefs como sobre los planes. Las modificaciones realizadas se han implementado en las funciones selectoras y en los procesos de unificación. El código de G-JASON está disponible en `https://github.com/secharte/ab/`. Se presentan los cambios propuestos según el orden en el ciclo de interpretación de un programa de agente, siguiendo el flujo de ejecución en la máquina de estados (Figura 2).

Selección de eventos: durante la ejecución del proceso de inicialización de la máquina de estados se cargan el conjunto de eventos y el conjunto de todos los planes. El conjunto de eventos se ha generado a partir de las creencias definidas para el agente y la librería de planes se ha generado con todos los planes que contiene el agente.

1. *Opción utilizando grados en las creencias:* las creencias generan los eventos que luego unificarán con los eventos disparadores de los planes. Se decidió que la "selectora de eventos" seleccione el evento con mayor grado de certeza (degOfCert), modelando atender primero los hechos más confiables. El método recibe como parámetro una cola de eventos pero retorna el evento que posee el mayor grado de certeza (defOfCert).
   Si se generaron los eventos:
   $E1 = [+sol[degOfCert(0.8)]]$ y $E2 = [+viento[degOfCert(0.7)]]$
   La selectora retorna el evento $E1$ debido a que es el evento con mayor grado de certeza.
2. *Activando prioridades:* para manejar la reactividad del agente de una forma más directa, se agrega la posibilidad de activar un archivo "'*Prioridades*"' donde se ordenan los eventos según su prioridad. Luego, cuando esta prioridad es mayor, los eventos serán más reactivos dado que la selectora de eventos los considerará primero. Para el tratamiento de este archivo se ha modificado el proceso de selección de eventos nuevamente y se determinará en estas propiedades el orden en el cual se seleccionarán los eventos, quedando sin efecto en este caso la selección de eventos observada anteriormente.
   Por ejemplo, en una situación donde se requiere establecer el orden de importancia de tres hechos gas, luz, vibración, donde la importancia de los hechos a tratar sigue ese orden, se establece este orden de precedencia en el archivo. *Prioridades*.

Proceso de unificación de eventos: se procede a unificar el evento seleccionado por la selectora de eventos contra los eventos disparadores contenidos en los planes de la librería de planes. Si se tiene un *Plan P*:

$@PlanP + sensor\_termico(TEMP)[degOfCert(0.7)] : (TEMP < 300) < -encender\_alarma; llamar\_supervisor$,

y un evento : $E = [+sensor\_termico(100)[degOfCert(0.8)]]$:

El evento E y el plan P unifican según la unificación original de JASON: sensor_termico(100) del evento unifica con sensor_termico(TEMP) del plan y quedará TEMP=100.

Se extendió esta unificación y se seleccionarán los planes donde además, el grado del evento unificado sea mayor al grado del evento disparador del plan considerado.

En el ejemplo, el grado del evento E es 0.8 y el grado del evento disparador del $PlanP$ es 0.7. Luego este plan será considerado relevante.

Proceso de unificación de los contextos el proceso de unificación de eventos arroja un conjunto de planes relevantes los cuales serán evaluados en el proceso de unificación de contexto. Originalmente en JASON se evalúa que el contexto sea verdadero. Por ejemplo, si se tiene que el *planP* (citado anteriormente) es relevante y una creencia en la base: $B = [+sensor\_termico(100)[degOfCert(0.8)]]$:

Como observamos en la unificación anterior TEMP tomara el valor 100, con lo cual la validación del contexto (TEMP <300) será verdadera. Se extendió el proceso de unificación de contexto y se seleccionará el plan relevante donde el grado del belief en la base sea mayor al grado del contexto en el plan. En el ejemplo, el grado del belief B es 0.8 y el grado del contexto del plan relevante P es 0.6 por lo tanto el $PlanP$ se considerará *aplicable*.

Selección de opciones: la unificación de contextos arrojará un conjunto de planes aplicables, la selectora de opciones elige uno de estos planes aplicables y luego vincula este plan con alguna intención existente (en caso de el evento fue interno, generado a partir de una creencia) o se creará una nueva intención que contenga esta opción con su respectivo plan asociado. Las opciones estarán representadas por los planes aplicables asociados. En G-JASON se modificó el método "selectora de opciones" ("*selectOption*") para que seleccione la opción con mayor relevancia (planRelevance). Luego, si se presentan las opciones:

$P1 = @Plan1[planRelevance(0.85)]+sensor\_termico(TEMP)[degOfCert(0.7)] : (TEMP < 300)[degOfCert(0.6)] < -encender\_alarma; llamar\_supervisor,$

$P2 = @Plan2[planRelevance(0.65)]+sensor\_termico(TEMP)[degOfCert(0.5)] : (TEMP < 100)[degOfCert(0.3)] < -encender\_calentador; encender\_turbina,$

La selectora retorna la opción P1 debido a que es la opción (plan) con mayor relevancia (planRelevance).

Selección de intenciones: en la selección de opciones del proceso se crearon o actualizaron el conjunto de intenciones que posee el agente. El conjunto de intenciones será la entrada del método de selección de intenciones, donde se eligirá una intención a ejecutar para terminar el ciclo de interpretación del agente. Cada intención estará representada por el plan que cada una de ellas contiene y el cuerpo de ese plan contiene una lista de acciones para su posterior ejecución. El orden de ejecución de acciones de una intención se mantiene según la versión original de JASON. Al igual que en la selección de opciones, la selectora de intenciones tendrá en cuenta el valor del grado en los planes (*planRelevance*) y se seleccionará la intención con mayor grado de relevancia (planRelevance).Una vez seleccionada la intención, la máquina de estados ejecuta una de las acciones contenidas en el cuerpo del plan.

5 Caso de Estudio

Se presenta un caso de estudio que muestra el potencial de las extensiones anteriormente descriptas. Se desea modelar la supervisión de un horno rotativo utilizado para la fundición de metales. Se analizan constantemente tres variables fundamentales en la operación: temperatura, presión y vibración. Para esta tarea se cuenta

con tres sensores estrategicamente situados. Por lo tanto, los sensores proveen al agente de información sobre las lecturas y sus grados de certeza asociados los cuales dependerán de la precisión de los instrumentos utilizados. El agente además posee acciones recomendadas de acuerdo a las lecturas de los sensores, las cuales serán modeladas dentro de los planes. Para el caso de la variable presión por ejemplo, si la lectura supera los 35 bars se deberá cerrar una válvula de inyección del horno (plan P3) y si la presión supera los 70 bars se deberá encender una alarma general, dado el peligro de explosión en el horno y se deberán tomar medidas de precaución (plan P4).

Se sabe además, que la precisión de los sensores de temperatura son de 70% a 700 grados por lo tanto el grado de certeza de la lectura de temperatura (B1) será de 0.7. Para los planes se agregarán grados (planRelevance) para dar relevancia a la necesidad de urgencia de ejecutar ciertas acciones por el agente supervisor. Dada la relevancia de la presión para los hornos, se agregan los grados de relevancia más altos a los planes relacionados a la presión P3 (0,9) y P4 (0,85) respectivamente.

B1= $sensor\_termico(700)[degOfCert(0.7)]$

B2= $sensor\_presion(80)[degOfCert(0.9]$.

B3= $sensor\_vibracion(8)[degOfCert(0.6)]$.

P1= $@alerta\_temperatura[planRelevance(0.7)]$

$+sensor\_termico(TEMP)[degOfCert(0.5)]$:

$TEMP > 300[degOfCert(0.6)] < -prende\_ventilador$.

P2= $@urgencia\_temperatura[planRelevance(0.8)]$

$+sensor\_termico(TEMP)[degOfCert(0.5)]$:

$TEMP > 600[degOfCert(0.6)] < -$**apagar_horno**.

P3= $@alerta\_presion[planRelevance(0.85)]+sensor\_presion(PRES)[degOfCert(0.7)]$:

$PRES > 35[degOfCert(0.8)] < -cierra\_valvula$.

P4= $@urgencia\_presion[planRelevance(0.9)]+sensor\_presion(PRES)[degOfCert(0.7)]$:

$PRES > 70[degOfCert(0.8)] < -$**enciende_alarma**.

P5= $@manejo\_vibracion[planRelevance(0.6)]+sensor\_vibracion(NVL)[degOfCert(0.4)]$:

$NVL > 5[degOfCert(0.5)] < -$**frena_motor**.

Sin considerar un archivo de prioridades, el orden de las acciones resultantes a ejecutar por el agente son: **(1) enciende_alarma, (2) apaga_horno y (3)frena_motor**.

Se observa que la acción *enciende_alarma* es la primera en ejecutarse. Esta acción esta relacionada con el manejo de presión modelada por el belief B2 (grado de certeza 0,9) el cual es el más confiable. Además, de los planes aplicables (P3 y P4) se ha ejecutado primero el plan P4 (relevancia de 0,9), por ser el plan más relevante. Ejecutando este ejemplo en la versión de JASON original, no hay posibilidad de explicitar la certeza o prioridad, en las creencias o en los planes y se ejecutó primero la acción relacionada a la temperatura dado el orden de llegada de las lecturas de los sensores. El agente supervisor implementado en G-JASON mejora su performance al atender primero las lecturas más confiables (sensor de presión) y ejecutar las acciones vinculadas a este evento, en el orden más relevante.

6 Conclusiones

Se ha realizado una extensión de JASON tanto sintáctica como semántica para incluir grados en las creencias y en los planes (en distintos componentes). También se ha agregado el concepto de prioridades para poder ordenar los eventos según el orden de reactividad que se les quiera otorgar. Al incluir las prioridades y grados mencionados, para que los mismos tengan el valor semántico deseado, se tuvieron que modificar las funciones selectoras y los procesos de unificación para los eventos y los contextos de los planes. Se implementaron todas las modificaciones necesarias, en una nueva versión de JASON que denominamos G-JASON. A través de un caso de estudio se pudo comprobar que con esta extensión se aumenta la expresividad de JASON pudiendo representar una situación que involucra incertidumbre en los hechos y diferentes relevancias de planes obteniendo mejores resultados que en JASON original. Respecto al modelo BDI graduado que ha inspirado este trabajo ha quedado pendiente modelar grados en los deseos (goals de JASON) ya que en la estructura de los sistemas PRS estos elementos no son considerados en los componentes básicos y no tienen un representación adecuada. Se plantea como trabajo futuro, trabajar en la modelización de los grados de deseos del agente en estas plataformas.

References

1. Bordini, R., Hübner, J.: BDI Agent Programming in AgentSpeak Using JASON. John Wiley and Sons. (2007)
2. Casali, A., Godo, Ll., Sierra, C.: A graded BDI agent model to represent and reason about preferences: Artificial Intelligence, Special Issue on Preferences Artificial Intelligence. vol. 175, pp. 1468–1478. (may, 2011)
3. Casali, A., Godo, Ll., Sierra, C.: Lecture Notes in Artificial Intelligence: Graded BDI Models For Agent Architectures. Leite, Joao and Torroni, Paolo, Springer-Verlag. 126–143. Berling Heidelberg (2005)
4. Casali, A., Godo, Ll., Sierra, C.: A Tourism Recommender Agent: From theory to practice. In: Revista Iberoamericana de Inteligencia Articial, vol. 12:40, pp. 23–38. (2008)
5. D'Inverno, M., Kinny, D., Luck, M., Wooldridge, M.: A formal specification of dMARS. Intelligent Agents IV: Proc. Fourth International Workshop on Agent Theories, Architectures and Languages. Singh, M.P. and Rao, A.S. and Wooldridge, M. Springer-Verlag, 155–176, Montreal, (1998)
6. Georgeff, M. P., Lansky, A. L.: Reactive reasoning and planning. AAAI-87, 677–682, Seattle. (1987)
7. Georgeff, M., Pell, B., Pollack, M., Tambe, M., Wooldridge, M.: The Belief-Desire-Intention Model of Agency. Intelligent Agents. Muller, J. P. and Singh, M. and Rao, A. S. Springer-Verlag, vol. 1365, Berling Heidelberg. (1999)
8. Krapf, A., Casali, A.: Desarrollo de Sistemas Inteligentes aplicados a redes elctricas industriales. In: WASI-CACIC, Corrientes, Argentina (2007)
9. Rao, A., Georgeff, M.: BDI Agents from Theory to Practice. In: AAII. (1995)
10. Wooldridge, M.: Introduction to Multiagent Systems, 2 Ed., John Wiley and Sons, (2009).