Universidad Autónoma del Estado de Hidalgo

Instituto de Ciencias Básicas e Ingeniería

Área Académica de Computación y Electrónica

Comunicación entre Agentes

Nombres:

Guerrero Olvera Angélica

Semestre: 5° Grupo: 4

Introducción

Los agentes inteligentes se han vuelto muy importantes en la creación de sistemas que necesitan que diferentes componentes trabajen de manera autónoma y se adapten a distintos entornos. Para facilitar esto, la Fundación para Agentes Físicos Inteligentes (FIPA) ha establecido estándares que permiten a los agentes comunicarse y colaborar mejor entre ellos. Entre estos estándares se encuentran el FIPA ACL Message Structure y la FIPA Communicative Act Library, que establecen un lenguaje común para que los agentes intercambien información y se coordinen usando mensajes bien estructurados.

El FIPA ACL (Agent Communication Language) define cómo deben ser los mensajes que los agentes se envían, utilizando un conjunto de acciones predefinidas como inform, request, agree, y cancel. Estas acciones permiten a los agentes expresar claramente lo que quieren hacer, como compartir información, aceptar propuestas, solicitar tareas, o avisar que algo no salió bien. Gracias a estos documentos de FIPA, es posible diseñar sistemas donde los agentes, aunque sean creados por diferentes personas o empresas, puedan comunicarse sin problemas.

Por otro lado, la Guía de Programación de JADE explica cómo crear agentes y sus comportamientos utilizando JADE (Java Agent Development Framework), una plataforma que sigue los estándares de FIPA. JADE simplifica el trabajo de crear, gestionar y comunicar agentes mediante un conjunto de herramientas que hacen más fácil la comunicación entre ellos. Además, los comportamientos que se pueden configurar en JADE, como SimpleBehaviour, CyclicBehaviour y FSMBehaviour, permiten que los desarrolladores definan cómo deben reaccionar los agentes ante los mensajes que reciben.

Por eso, los documentos proporcionados ofrecen una base sólida para entender cómo los agentes pueden interactuar de forma autónoma y flexible, lo que facilita la creación de sistemas complejos en áreas como comercio electrónico, robótica, automatización industrial y sistemas de apoyo a la toma de decisiones.

COMUNICACIÓN ENTRE AGENTES

¿Cómo se comunican los agentes?

En JADE, los agentes se comunican usando un protocolo basado en el estándar FIPA ACL (Agent Communication Language). Los mensajes ACL permiten a los agentes intercambiar información de forma asincrónica mediante el uso de una estructura predefinida, que incluye los siguientes elementos clave:

Performative (Acto Comunicativo): Indica la intención del mensaje, como inform, request, agree, etc.

Sender y Receiver: Especifican quién envía y quién recibe el mensaje.

Content: Contenido o información que se transmite.

Ontology: Define el conjunto de términos que tanto el emisor como el receptor entienden.

Language y Encoding: Indican cómo interpretar el contenido del mensaje.

Protocol: Define el protocolo de interacción (por ejemplo, fipa-contract-net o fipa-request).

El uso de performatives estandarizados asegura que los agentes entiendan correctamente la intención detrás de cada mensaje

LENGUAJE DE COMUNICACION PARA AGENTES (ACL)

El Lenguaje de Comunicación para Agentes (ACL, por sus siglas en inglés: Agent Communication Language) es un sistema de comunicación diseñado para que los agentes de software (programas autónomos) puedan interactuar entre sí de manera efectiva. Este lenguaje es clave en sistemas multiagente, donde varios agentes trabajan juntos en tareas complejas o colaboran en la resolución de problemas.

Componentes clave del ACL

Actos de Comunicación (Speech Acts): En ACL, cada mensaje representa un "acto de comunicación". Estos actos están inspirados en la teoría de actos de habla y son instrucciones o peticiones específicas que un agente envía a otro. Algunos ejemplos comunes incluyen

- Informar (inform): el agente A comunica al agente B algún dato o hecho.

- Solicitar (request): el agente A solicita al agente B que realice una acción.

- Confirmar (confirm): el agente A verifica una información específica con el agente B.

- Proponer (propose): el agente A hace una propuesta de acción o colaboración al agente B.

Intenciones y Propósitos: ACL se centra en la intención detrás de cada mensaje, es decir, en el propósito que tiene el agente emisor al enviar una comunicación. Esta intención ayuda al receptor a interpretar correctamente el mensaje. Por ejemplo, si un agente envía un inform sobre una condición climática, su intención podría ser "notificar" para que otro agente se prepare.

Estructura del Mensaje ACL: Los mensajes en ACL tienen una estructura específica que incluye varios parámetros, como:

- Performativa: el tipo de acto de comunicación (como inform o request).

- Remitente: el agente que envía el mensaje.

- Receptor: el agente que recibe el mensaje.

- Contenido: la información que el remitente desea comunicar.

- Lenguaje: el lenguaje en el que está escrito el contenido.

- Contexto: el contexto en el que se envía el mensaje, que puede incluir información adicional sobre las creencias o estados del agente.

Esta estructura ayuda a garantizar que los mensajes sean precisos y que los agentes puedan procesarlos adecuadamente.

Protocolos de Comunicación

Para que los agentes entiendan cómo reaccionar a los mensajes, ACL también define protocolos de comunicación. Estos son reglas o secuencias de actos de comunicación que los agentes deben seguir en una interacción particular. Por ejemplo:

- Negociación: una secuencia de mensajes de propuesta y contraoferta hasta que los agentes lleguen a un acuerdo.

- Consulta y Respuesta: un agente solicita información, y el otro responde con la información requerida o una negativa si no tiene la información.

Ejemplo de Interacción con ACL

Imagina dos agentes, A y B, que colaboran en un sistema de compra de boletos. Un intercambio típico podría ser:

- A solicita a B: request - "Encuentra un boleto de avión a París."

- B responde: inform - "He encontrado un boleto de avión a París por $200."

- A confirma: confirm - "Reserva ese boleto."

- B confirma la acción: inform - "Boleto reservado."

Usos y Ventajas de ACL

El ACL es útil en sistemas donde los agentes tienen que colaborar o intercambiar información sin intervención humana directa. Algunos ejemplos son:

- Automatización de procesos: agentes que colaboran para controlar un sistema de fabricación.

- Comercio electrónico: agentes que negocian precios o realizan transacciones.

- Gestión de información: agentes que buscan, filtran y distribuyen información en una red.

Implementaciones de ACL

El ACL más conocido es el de la Fundación de FIPA (Fundación para la Promoción de la Investigación en Agentes), que establece estándares para las arquitecturas y lenguajes de agentes. FIPA ACL se usa como base en muchas plataformas de sistemas multiagente, como JADE (Java Agent DEvelopment Framework).

ESTRUCTURA DE UN MENSAJE ACL DE FIPA

La estructura de los mensajes ACL (Agent Communication Language) definida por FIPA (Foundation for Intelligent Physical Agents) es un estándar que especifica cómo los agentes deben intercambiar información en sistemas multi-agente para asegurar una comunicación efectiva y la interoperabilidad entre distintas plataformas de agentes.

Un mensaje ACL tiene una estructura bien definida que consta de varios parámetros obligatorios y opcionales que se utilizan para identificar el contenido y la intención del mensaje, así como para gestionar el control del flujo de conversación entre los agentes. A continuación, se presenta un análisis detallado de los principales componentes de un mensaje ACL según las especificaciones de FIPA:

1. Performative (Acto Comunicativo)

El parámetro performative es obligatorio y representa el tipo de acto comunicativo que el mensaje intenta realizar. FIPA define un conjunto estándar de performatives, como:

inform: El agente informa que una proposición es verdadera.

request: El agente solicita que otro agente realice una acción.

agree: Indica que un agente acepta realizar una acción solicitada.

cfp (call for proposal): El agente busca propuestas para una tarea.

accept-proposal: Acepta una propuesta previa.

reject-proposal: Rechaza una propuesta previa.

failure: Indica que la acción solicitada no se pudo completar.

cancel: El agente ya no tiene la intención de que se realice una acción solicitada.

El performative establece el contexto y la semántica del mensaje, guiando al receptor sobre cómo debe interpretar el contenido.

2. **Participantes en la Comunicación**

Estos parámetros identifican a los agentes que participan en la comunicación:

Sender (Remitente): El agente que envía el mensaje.

Receiver (Receptor): Uno o más agentes destinatarios del mensaje.

Reply-to: Indica a qué agente deben dirigirse las respuestas. Este parámetro es útil si el remitente desea que las respuestas se envíen a un agente diferente al que originalmente envió el mensaje.

Estos parámetros son cruciales para garantizar que los mensajes lleguen a los agentes correctos y que las respuestas se envíen de vuelta a los agentes esperados, especialmente en interacciones complejas.

3. Contenido del Mensaje

El parámetro content contiene la información o acción que se está comunicando. El contenido es expresado en un lenguaje que debe ser comprensible tanto para el remitente como para el receptor. Este parámetro puede incluir:

Proposiciones (como en el acto inform).

Expresiones de acciones (como en el acto request).

El contenido debe ser interpretado por el receptor en función del lenguaje y la ontología especificados.

4. Descripción del Contenido

Para interpretar correctamente el contenido del mensaje, se deben definir los siguientes parámetros:

Language (Lenguaje): Define el lenguaje formal en el que está expresado el contenido (por ejemplo, FIPA-SL, XML).

Ontology (Ontología): Describe el conjunto de conceptos y relaciones que se utilizan en el contenido. Asegura que ambos agentes compartan un entendimiento común sobre los términos usados en el mensaje.

Encoding (Codificación): Especifica la forma en que el contenido está codificado (por ejemplo, UTF-8).

Estos parámetros garantizan que los agentes comprendan el significado del contenido más allá del simple texto, ya que se basan en un conjunto de conceptos predefinidos.

5. Control de la Conversación

Para gestionar el flujo de la conversación entre los agentes, se utilizan varios parámetros de control:

Protocol: Define el protocolo de interacción que rige la conversación (por ejemplo, fipa-request, fipa-contract-net).

Conversation-ID: Un identificador único para rastrear una conversación particular. Esto es útil en escenarios donde múltiples conversaciones ocurren simultáneamente.

Reply-with: Indica un identificador que el receptor puede usar al responder al mensaje.

In-reply-to: Utilizado para correlacionar un mensaje de respuesta con el mensaje original al que está respondiendo.

Reply-by: Especifica un plazo de tiempo para que el receptor responda al mensaje.

ACTOS COMUNICATIVOS DE FIPA

Los actos comunicativos son acciones realizadas por los agentes para comunicarse entre sí. FIPA (Foundation for Intelligent Physical Agents) ha definido un conjunto de actos comunicativos estándar en el lenguaje ACL:

Refuse: Se utiliza cuando un agente rechaza realizar una acción que se le ha solicitado. El agente emisor indica que no tiene la intención de llevar a cabo la acción solicitada.

Ejemplo: Un agente A solicita a otro agente B que realice una tarea, pero B responde con un mensaje refuse indicando que no puede o no desea realizarla.

Reject Proposal: Este acto se emplea para rechazar una propuesta que fue previamente enviada por otro agente. Generalmente, se utiliza en protocolos de negociación donde se intercambian propuestas y contrapropuestas.

Ejemplo: Si un agente realiza una propuesta para vender un producto y el otro agente no está de acuerdo con los términos, responderá con reject-proposal.

Request: Se usa para solicitar a un agente que realice una acción específica. El agente que recibe el mensaje request debe interpretar la solicitud y decidir si la acepta o la rechaza.

Ejemplo: Un agente envía un mensaje request para que otro agente le envíe información sobre un recurso específico.

Request When: Similar al acto request, pero la acción solicitada solo se debe realizar cuando una condición específica se cumpla.

Ejemplo: Un agente podría enviar un request-when solicitando la actualización de un informe solo cuando un evento específico ocurra.

Request Whenever: Es una extensión del request when, donde la solicitud se repite cada vez que se cumple la condición dada.

Ejemplo: Un agente puede usar request-whenever para ser notificado cada vez que cambie el estado de un senso.

Subscribe: Este acto es utilizado por un agente para suscribirse a cambios o actualizaciones de información de otro agente. El agente receptor enviará actualizaciones al suscriptor siempre que ocurra un cambio relevante.

Ejemplo: Un agente se suscribe para recibir notificaciones sobre el precio de un activo financiero

ACTOS COMUNICATIVOS

Accept Proposal

Descripción: Este acto se utiliza para aceptar una propuesta que otro agente ha hecho anteriormente. Por ejemplo, en un protocolo de negociación, un agente puede enviar un cfp (call for proposal) a varios agentes solicitando propuestas, y luego utilizar accept-proposal para aceptar una de ellas.

Formalización: <i, accept-proposal(j, <j, act>, φ)> significa que el agente i acepta que j realice la acción act bajo las condiciones φ.

Ejemplo:

El agente Comprador envía un cfp a varios agentes Vendedores solicitando cotizaciones para un producto.

El Vendedor A responde con una oferta.

Si el Comprador acepta la oferta, enviará un accept-proposal al Vendedor A confirmando su interés en comprar al precio ofrecido.

Agree

Descripción: El acto agree indica que un agente acepta realizar una acción que otro agente le ha solicitado, pero no necesariamente de inmediato. Puede haber condiciones adicionales que deben cumplirse antes de ejecutar la acción.

Formalización: <i, agree(j, <i, act>, φ)> significa que el agente i está de acuerdo en realizar la acción act para j cuando se cumplan las condiciones φ.

Ejemplo:

El agente Cliente solicita a un Proveedor que entregue un paquete.

El Proveedor responde con agree indicando que lo hará cuando disponga de un vehículo de entrega disponible.

Cancel

Descripción: cancel se utiliza para informar que un agente ya no tiene la intención de que se realice una acción previamente solicitada. Esto no implica necesariamente que la acción se detenga, sino simplemente que el agente ya no tiene interés en ella.

Formalización: <i, cancel(j, a)> indica que i informa a j que ya no tiene la intención de que j realice la acción a.

Ejemplo:

Si un Cliente solicitó un servicio a un Proveedor pero decide cancelarlo antes de que sea iniciado, envía un cancel.

Call for Proposal (CFP)

Descripción: Un cfp se utiliza para iniciar un proceso de negociación. Un agente solicita propuestas de otros agentes para llevar a cabo una tarea. El mensaje puede contener restricciones o condiciones que deben cumplirse en la propuesta.

Formalización: <i, cfp(j, <j, act>, Ref x φ(x))> indica que el agente i solicita a j que proponga cómo realizar la acción act bajo ciertas condiciones.

Ejemplo:

El agente Organizador de Eventos envía un cfp a varios agentes para recibir ofertas de catering para un evento.

Confirm

Descripción: confirm se utiliza para afirmar que una proposición es verdadera, especialmente cuando el receptor puede tener dudas sobre la información.

Formalización: <i, confirm(j, φ)> significa que el agente i informa a j que cree que la proposición φ es verdadera.

Ejemplo:

El agente Sensor confirma a un agente de Monitoreo que la temperatura ha alcanzado el umbral crítico.

Disconfirm

Descripción: Similar a confirm, pero en este caso, el agente informa que una proposición es falsa, especialmente si el receptor podría creer lo contrario.

Formalización: <i, disconfirm(j, φ)>.

Ejemplo:

El agente Control de Inventario desmiente que haya stock disponible para un producto solicitado.

Failure

Descripción: Se utiliza para informar que una acción solicitada no pudo completarse debido a algún fallo. El mensaje puede incluir la razón del fallo.

Formalización: <i, failure(j, a, φ)> indica que i no pudo realizar la acción a para j debido a la condición φ.

Ejemplo:

El agente Servidor informa a un Cliente que no pudo procesar su solicitud debido a un error de red.

Inform

Descripción: inform indica que un agente comunica que una proposición es verdadera. El emisor espera que el receptor actualice sus creencias basándose en esta información.

Formalización: <i, inform(j, φ)> donde φ es la proposición que se cree verdadera.

Ejemplo:

Un Sensor de Movimiento informa al sistema de seguridad que se detectó movimiento en una zona.

INFORM IF

Contenido del mensaje: Una proposición.

Definición

La macroacción **"inform if"** es una abreviatura para indicar si se cree o no en una proposición dada. El agente que ejecuta esta macroacción cuando realmente realizará una acción estándar de informar.

El contenido del acto de informar dependerá de las creencias del agente que informa. Para realizar una acción informar-si sobre una proposición cerrada φ:

- Si el agente cree en la proposición, informará al otro agente que φ es verdadera.
- Si cree en la negación de la proposición, informará que φ es falsa, es decir, ¬φ.

En otras circunstancias, puede que el agente no pueda llevar a cabo este plan. Por ejemplo, si no tiene conocimiento de φ o no permite que la otra parte sepa (que cree) en φ, enviará un mensaje de rechazo.

Modelo formal

<i, inform-if (j, φ)> ≡

<i, inform (j, φ)>|<i, inform (j, ¬φ)> FP:

Bifi φ ∧ ¬B_i (Bifj φ ∨ Uifj φ) RE: Bifj

φ

Inform If, representa las dos posibles acciones a realizar:

- i informa a j que φ

- i informa a j que no φ

INFORM REF

Contenido del mensaje: Una descripción del objeto (una expresión referencial).

Definición

La macroacción **"inform-ref"** permite al emisor informar al receptor de algún objeto que el emisor cree que corresponde a un descriptor, como un nombre u otra descripción que la identifique.

Inform-ref es una macroacción, ya que corresponde a una disyunción (posiblemente infinita) de actos inform.

El agente que realiza el acto debe creer que el objeto o conjunto de objetos correspondiente a la expresión de referencia es el suministrado, y no debe creer que el

receptor del acto ya sabe qué objeto o conjunto de objetos corresponde a la expresión de referencia. El agente puede optar por enviar un mensaje de rechazo si no es capaz de establecer las condiciones previas del acto.

Modelo formal

<i, inform-ref (j, Ref x δ(x))> ≡

<i, inform (j, Ref x δ(x) = r_1)> | ... |

(<i, inform (j, Ref x δ(x) = rk)>

FP: $Bref_i$ Ref x δ(x) ∧ ¬B_i($Bref_j$ Ref x δ(x) ∨ $Uref_j$ Ref x δ(x))

RE: $Bref_j$ Ref x δ(x)

NOT UNDERSTOOD

Contenido del mensaje: Una tupla formada por una acción o acontecimiento, por ejemplo, un acto comunicativo, y una razón explicativa.

Definición

El emisor del acto comunicativo “not understood” recibió un acto comunicativo que no entendió. Esto puede deberse a varias razones: el agente puede no haber sido diseñado para procesar un determinado acto o clase de actos, o puede haber estado esperando un mensaje diferente. Este acto también puede utilizarse en el caso general para que i informe a j de que no ha entendido la acción de j.

El segundo elemento de la tupla de contenido del mensaje es una proposición que representa la razón del fallo de comprensión. No hay garantía de que la razón se represente de forma que el agente receptor la entienda. Sin embargo, un agente cooperativo intentará explicar el malentendido de forma constructiva.

No es posible capturar completamente la semántica pretendida de una acción no entendida por otro agente.

φ debe ser una fórmula bien formada del lenguaje de contenido del agente emisor. Si el emisor utiliza el mensaje textual desnudo, es decir, cadena en la definición de sintaxis, como la razón φ, debe ser una afirmación proposicional y (al menos) el emisor puede entender ese mensaje (en lenguaje natural) y calcular su valor de verdad, es decir, decidir si su afirmación es verdadera o falsa.

Modelo formal

<i, not-understood(j, a, φ)> ≡

<i, inform(j, α) >

FP: Bi α ∧ ¬Bi (Bif_j α ∨ Uif_j α) RE:

Bj α

PROPAGATE

Contenido del mensaje: Una tupla de un descriptor, un acto comunicativo ACL incrustado y una condición de restricción para la propagación, por ejemplo, un tiempo de espera.

Definición
Se trata de una acción compuesta de las dos acciones siguientes:

- El agente emisor solicita al receptor que trate el mensaje incrustado en el mensaje de propagación recibido como si fuera enviado directamente desde el emisor, es decir, como si el emisor realizara el acto comunicativo incrustado directamente al receptor.

- El emisor desea que el receptor identifique los agentes denotados por el descriptor dado y les envíe una versión modificada del mensaje propagado recibido, como se describe a continuación:

En el reenvío, el parámetro receptor del mensaje de propagación reenviado se establece en el agente y el parámetro emisor se establece en el receptor del mensaje de propagación recibido.

El emisor y el receptor del acto comunicativo integrado del mensaje de propagación reenviado también se establecen en el mismo agente que el emisor y el receptor del mensaje de propagación reenviado, respectivamente.

Este acto comunicativo está diseñado para entregar mensajes a través de agentes federados creando una cadena (o árbol) de mensajes propagados.

Modelo formal

$\langle i, \text{propagate } (j, \text{Ref } x\ \delta(x), \langle i, \text{cact}\rangle, \varphi)\rangle \equiv$

$\langle i, \text{cact}(j)\rangle$;

$\langle i, \text{inform } (j, I_i((\exists y)(Bj\ (\text{Ref } x\ \delta(x) = y) \wedge$

Done $(\langle j, \text{propagate } (y, \text{Ref } x\ \delta(x), \langle j, \text{cact}\rangle, \varphi)\rangle, Bj\ \varphi))))\rangle$ FP:

FP (cact) $\wedge$ Bi α $\wedge$ ¬Bi (Bifj α $\vee$ Uifj α)

RE: Done (cact) $\wedge$ Bj α

PROPOSE

Contenido del mensaje: Una tupla que contiene una descripción de la acción, que representa la acción que el remitente propone realizar, y una proposición que representa las condiciones previas para la realización de la acción. y una proposición que representa las condiciones previas a la ejecución de la acción.

Definición

“Propose” es un acto de propósito general que consiste en hacer una propuesta o responder a una propuesta existente durante un proceso de negociación, proponiendo realizar una acción determinada siempre que se cumplan determinadas condiciones.

El emisor de la propuesta informa al receptor de que el proponente adoptará la intención de realizar la acción una vez que se cumpla la condición previa dada, y el receptor notifica al proponente la intención del receptor de que el proponente realice la acción.

Modelo formal

<i, propose (j, <i, act>, φ)> ≡

<i, inform (j, I_j Done (<i, act>, φ) I_i Done (<i, act>, φ))>

FP: Bi α ∧ ¬Bi (Bifj α ∨ Uifj α)

RE: B_j α

PROXY

Contenido del mensaje: Una expresión referencial, que denota los agentes de destino, un mensaje ACL, que se realizará a los agentes, y una condición de restricción para realizar el acto comunicativo incrustado, por ejemplo, el número máximo de agentes que se reenviarán, etc.

Definición

El agente emisor informa al receptor de que el emisor desea que el receptor identifique los agentes que satisfacen el descriptor dado y que les realice el acto comunicativo incrustado, es decir, el receptor les envía el mensaje incrustado.

Al realizar el acto comunicativo incrustado, el parámetro receptor se establece en el agente denotado y el emisor se establece en el receptor del mensaje proxy. Si el acto comunicativo incrustado contiene un parámetro de respuesta.

En el caso de una solicitud de intermediación (es decir, el parámetro de protocolo se establece en fipa-brokering), el agente de intermediación (el receptor del mensaje proxy) debe registrar algunos parámetros.

Modelo formal

<i, proxy (j, Ref x δ(x), <j, cact>, φ)> ≡

<i, inform (j, I_i((∃ y)(Bj (Ref x δ(x) = y) ∧

Done (<j, cact(y)>, Bj φ))))>

FP: Bi α ∧ ¬Bi (Bifj α ∨ Uifj α)

RE: Bj α

QUERY IF

Contenido del mensaje: Una proposición

Definición

“Query if” es el acto de preguntar a otro agente si (cree que) una determinada proposición es verdadera. El agente emisor solicita al receptor que le informe de la veracidad de la proposición.

El agente que realiza el acto query-if:

- No conoce el valor de verdad de la proposición.

- Cree que el otro agente puede informar al agente que realiza la consulta si conoce la verdad de la proposición.

Modelo formal

<i, query-if (j, φ)> ≡

<i, request (j, <j, inform-if (i, φ)>)>

FP: $\neg Bif_i\, \varphi \wedge \neg Uif_i\, \varphi \wedge \neg B_i I_j$ Done(<j, inform-if (i, φ)>) RE:

Done (<j, inform(i, φ)>|<j, inform (i, ¬φ)>)

QUERY REF

Contenido del mensaje: Un descriptor (una expresión referencial).

Definición

“query-ref” es el acto de pedir a otro agente que informe al solicitante del objeto identificado por un descriptor. El agente emisor está solicitando al receptor que realice un acto inform, que contenga el objeto que corresponde al descriptor.

El agente que realiza el acto query-ref:

- No sabe qué objeto o conjunto de objetos corresponde al descriptor.

- Cree que el otro agente puede informar al agente que realiza la consulta del objeto o conjunto de objetos que corresponden al descriptor.

Modelo formal

<i, query-ref (j, Ref x δ(x))> ≡

<i, request (j, <j, inform-ref (i, Ref x δ(x))>)>

FP: $\neg Bref_i\, (Ref\ x\ \delta(x)) \wedge \neg Uref_i\, (Ref\ x\ \delta(x)) \wedge$

$\neg B_i I_j$ Done(<j, inform-ref (i, Ref x δ(x))>) RE:

Done(<i, inform (j, Ref x δ(x) = r1)> |...|

<i, inform (j, Ref x δ(x) = rk)>)

Refuse

Definición: El acto refuse se utiliza cuando un agente no acepta realizar la acción que se le ha solicitado. Este mensaje indica que el agente emisor no tiene la capacidad o el deseo de ejecutar la tarea.

Formalismo:

El agente i informa al agente j que no tiene la intención de realizar la acción solicitada.

El mensaje refuse no implica que el agente tenga que explicar la razón de su negativa, aunque puede hacerlo.

Uso típico: Se utiliza en escenarios de negociación donde un agente no puede o no desea cumplir con la solicitud de otro agente.

Ejemplo en ACL:

(refuse

:sender (agent-identifier :name i)

:receiver (set (agent-identifier :name j))

:content "Action not feasible"

:language fipa-sl)

Reject Proposal

Definición: El acto reject-proposal se utiliza cuando un agente rechaza una propuesta que ha recibido. Es común en protocolos de negociación como FIPA-Contract-Net.

Formalismo:

El agente i comunica al agente j que no acepta la propuesta que j había presentado previamente.

Uso típico: Se utiliza para rechazar propuestas en procesos de negociación después de haber evaluado las opciones disponibles.

Ejemplo en ACL:

(reject-proposal

:sender (agent-identifier :name i)

:receiver (set (agent-identifier :name j))

:in-reply-to proposal123)

Request

Definición: El acto request permite a un agente solicitar a otro que realice una acción específica.

Formalismo:

El agente i envía una solicitud al agente j para que ejecute una acción específica.

El agente receptor puede responder con agree, refuse o failure.

Uso típico: En situaciones donde un agente necesita delegar tareas a otros agentes.

Ejemplo en ACL:

```
(request
  :sender (agent-identifier :name i)
  :receiver (set (agent-identifier :name j))
  :content "Perform task"
  :language fipa-sl)
```

Request When

Definición: Similar al request, pero la acción solo se lleva a cabo cuando se cumple una condición específica.

Formalismo:

El agente i solicita al agente j que realice una acción solo cuando una condición predefinida se cumpla.

Uso típico: Para situaciones donde una acción depende de un evento o condición futura.

Ejemplo en ACL:

```
(request-when
  :sender (agent-identifier :name i)
  :receiver (set (agent-identifier :name j))
  :content "Action when condition is met"
  :language fipa-sl)
```

Request Whenever

Definición: Una extensión de request when, donde la acción se ejecuta repetidamente cada vez que se cumple la condición.

Formalismo:

El agente i solicita al agente j que realice una acción siempre que la condición específica se cumpla.

Uso típico: Utilizado en sistemas de monitoreo y notificación.

Ejemplo en ACL:

(request-whenever

:sender (agent-identifier :name i)

:receiver (set (agent-identifier :name j))

:content "Notify when condition changes"

:language fipa-sl)

Subscribe

Definición: El acto subscribe permite a un agente suscribirse a actualizaciones periódicas de otro agente sobre un tema específico.

Formalismo:

El agente i se suscribe a las actualizaciones del agente j respecto a un tema o recurso.

Uso típico: Común en sistemas de notificación en tiempo real donde un agente necesita estar informado de cambios.

Ejemplo en ACL:

(subscribe

:sender (agent-identifier :name i)

:receiver (set (agent-identifier :name j))

:content "Subscribe to updates"

:language fipa-sl)

Comportamientos de Agentes en JADE

Behaviour

Definición: Behaviour es la clase base para todos los comportamientos en JADE. Representa cualquier actividad que puede ser ejecutada por un agente.

Características:

Es una clase abstracta, lo que significa que se debe extender para definir comportamientos específicos.

Puede ser simple o compuesto, y se puede programar para que termine cuando se cumplan ciertas condiciones.

Métodos Clave:

action(): Define la lógica que se ejecuta cuando el comportamiento es seleccionado para ser ejecutado.

done(): Devuelve true si el comportamiento ha completado su tarea.

Ejemplo:

```
public class MyBehaviour extends Behaviour {
    public void action() {
        System.out.println("Performing  action...");
    }
    public boolean done() {
        return true; // Termina después de una sola ejecución
    }
}
```

SimpleBehaviour

Definición: SimpleBehaviour es una clase que extiende Behaviour y se utiliza para tareas que se ejecutan una sola vez o tienen una ejecución muy simple.

Características:

Es útil para comportamientos que no requieren ciclos complejos ni interacciones prolongadas.

Ideal para tareas únicas, como enviar un mensaje o realizar una consulta rápida.

Métodos Clave:

Hereda los métodos action() y done() de la clase Behaviour.

Ejemplo:

```
public class SendMessageBehaviour extends SimpleBehaviour {
    private boolean finished = false;
    public void action() {
        ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
        msg.addReceiver(new AID("receiver", AID.ISLOCALNAME));
        msg.setContent("Hello, this is a test message.");
        myAgent.send(msg);
        finished = true;
    }
    public boolean done() {
        return finished;
    }
```

}

OneShotBehavoiur

Un one-shot behaviour es una versión especializada de un comportamiento simple en el que el *método action() se ejecuta solo una vez. La clase jade.core.behaviours.OneShotBehaviour ya implementa el método done() devolviendo true, por lo que, al extenderla, la acción se ejecutará una única vez y luego se completará .*

Este tipo de comportamiento es útil para tareas que solo necesitan realizarse una vez y no requieren monitoreo continuo, como enviar un mensaje inicial, establecer un valor, o realizar una única consulta. En aplicaciones prácticas, es común utilizar OneShotBehaviour para inicializaciones o acciones que no deben repetirse, manteniendo así la simplicidad y eficiencia del agente.

```
public class MyOneShotBehaviour extends OneShotBehaviour {
  public void action() {
    // perform operation X
  }
}
```

CyclicBehaviour:

Es un comportamiento que nunca se completa y su método action() ejecuta las mismas operaciones cada vez que es invocado, En JADE, CyclicBehaviour implementa done() devolviendo false, permitiendo que la acción se repita continuamente hasta que el agente se termine .

Este tipo de comportamiento es ideal para tareas que requieren verificación constante o espera de un evento externo, como la recepción continua de mensajes en un entorno donde las interacciones entre agentes son impredecibles. Además, puede combinarse con otros comportamientos para asegurar que un agente responda en tiempo real a eventos específicos mientras realiza otras tareas

```
public class MyCyclicBehaviour extends CyclicBehaviour {
  public void action() {
    // perform operation Y
  }
}
```

CompositeBehaviour

La clase abstracta CompositeBehaviour en JADE permite crear comportamientos compuestos a partir de otros comportamientos (hijos). No define operaciones específicas ni una política de ejecución; esta última debe ser determinada por subclases como SequentialBehaviour, ParallelBehaviour y FSMBehaviour. Es una práctica recomendada utilizar subclases de CompositeBehaviour, a menos que se necesite una política de planificación especial para los comportamientos hijos.

CompositeBehaviour facilita el manejo de comportamientos complejos al organizar tareas menores de manera modular, y es especialmente valiosa en agentes que deben manejar

múltiples subtareas interdependientes, permitiendo una estructura clara y fácil de adaptar a diferentes necesidades de planificación.

SequentialBehaviour

Esta clase es un CompositeBehaviour que ejecuta sus subcomportamientos de forma secuencial y finaliza cuando todos los subcomportamientos se han completado. Se utiliza cuando una tarea compleja se pueda expresar como una secuencia de pasos atómicos (por ejemplo, realizar un cálculo, luego recibir un mensaje y luego realizar otro cálculo).

SequentialBehaviour garantiza que los subcomportamientos se ejecuten en un orden específico, lo cual es útil en flujos de trabajo lineales que dependen del resultado de cada paso previo. Su aplicación en el desarrollo de agentes es común en casos donde el agente debe completar una serie de operaciones con un orden bien definido, como una serie de interacciones en un protocolo de negociación.

ParallelBehaviour

Esta clase es un CompositeBehaviour que ejecuta sus subcomportamientos de manera concurrente y finaliza cuando se cumple una condición particular en sus subcomportamientos. Se proporcionan las constantes adecuadas que se deben indicar en el constructor de esta clase para crear un ParallelBehaviour que finaliza cuando se realizan todos sus subcomportamientos, cuando finaliza cualquiera de sus subcomportamientos o cuando finaliza un número N de sus subcomportamientos definido por el usuario. Utilice esta clase cuando una tarea compleja se pueda expresar como una colección de operaciones alternativas paralelas, con algún tipo de condición de finalización en las subtareas generadas.

En aplicaciones prácticas, ParallelBehaviour permite que un agente realice múltiples tareas simultáneamente, lo cual es esencial para sistemas en tiempo real o en entornos donde la rapidez de respuesta es clave, como en la colaboración de agentes en un entorno de tráfico inteligente.

FSMBBehavoiur

Esta clase es un CompositeBehaviour que ejecuta sus hijos de acuerdo con una máquina de estados finitos definida por el usuario. Más detalladamente, cada hijo representa la actividad que se realizará dentro de un estado de la FSM y el usuario puede definir las transiciones entre los estados de la FSM. Cuando el hijo correspondiente al estado Si se completa, su valor de terminación (tal como lo devuelve el método onEnd()) se utiliza para seleccionar la transición a ejecutar y se alcanza un nuevo estado Sj . En la siguiente ronda, se ejecutará el hijo correspondiente a Sj . Algunos de los hijos de un FSMBehaviour se pueden registrar como estados finales. El FSMBehaviour termina después de la finalización de uno de estos hijos.

Este comportamiento es particularmente útil para modelar agentes con flujos de trabajo complejos, donde la ejecución depende de transiciones entre diferentes estados de operación, como un sistema de gestión de procesos de negocio o agentes que interactúan en protocolos de comunicación con múltiples etapas.

WakerBehaviour

Ejecuta una acción única después de un tiempo de espera determinado. Tras el tiempo especificado, se invoca el método handleElapsedTimeout() y el comportamiento finaliza

WakerBehaviour es adecuado para tareas que deben ocurrir tras un cierto retraso sin que el agente quede inactivo en espera, como enviar un recordatorio o desencadenar un evento en un momento preciso. En entornos donde los agentes deben realizar tareas específicas en tiempos predeterminados, WakerBehaviour proporciona una forma efectiva de coordinar estas acciones sin bloquear otros comportamientos que el agente pueda estar realizando.

```
public class MyAgent extends Agent {
  protected void setup() {
    System.out.println("Adding waker behaviour");
    addBehaviour(new WakerBehaviour(this, 10000) {
      protected void handleElapsedTimeout() {
        // perform operation X
      }
    } );
  }
}
```

TickerBehaviour

Los métodos action() y done() ya están implementados de tal manera que ejecutan el método abstracto onTick() de manera repetitiva esperando un período determinado (especificado en el constructor) después de cada ejecución. Un TickerBehaviour nunca se completa.

Este comportamiento es útil en aplicaciones donde el agente debe realizar una tarea periódicamente, como verificar la disponibilidad de un recurso o actualizar su estado en intervalos específicos. TickerBehaviour permite al agente actuar en intervalos regulares sin bloquearse, lo que es ideal en sistemas donde se necesita una respuesta continua sin esperar eventos externos.

```
public class MyAgent extends Agent {
  protected void setup() {
    System.out.println("Adding waker behaviour");
    addBehaviour(new WakerBehaviour(this, 10000) {
      protected void handleElapsedTimeout() {
        // perform operation X
      }
    } );
  }
}
```

CONCLUSIONES

Los estándares de comunicación que propone FIPA para los agentes, a través del lenguaje ACL y su conjunto de acciones comunicativas, son muy importantes para que los agentes de diferentes desarrolladores y plataformas puedan interactuar sin problemas. Estos estándares no solo facilitan que los agentes trabajen juntos, sino que también hacen que los sistemas sean más eficientes y puedan crecer más fácilmente, ya que definen claramente cómo deben intercambiar información y coordinar sus acciones.

El FIPA ACL Message Structure establece un esquema claro para crear mensajes, definiendo elementos como el tipo de acción, el contenido, los participantes y los protocolos de comunicación. Esto garantiza que los agentes tengan un lenguaje común que puedan entender sin confusión, sin importar dónde o cómo se hayan creado. Gracias a actos comunicativos como call for proposal (llamado para propuestas), accept-proposal (aceptar propuesta), agree (acordar) e inform (informar), los agentes pueden negociar, colaborar y coordinarse para resolver problemas complejos y tomar decisiones de forma autónoma.

Por otro lado, la guía de programación de JADE brinda herramientas prácticas para que los desarrolladores apliquen estos estándares en proyectos reales. JADE permite diseñar agentes que pueden adaptarse a los cambios en su entorno y responder a diferentes situaciones de forma flexible y eficiente, gracias a su modelo basado en comportamientos.

Así que, al usar estándares como los de FIPA y plataformas como JADE, los desarrolladores pueden crear sistemas multi-agente que sean más sólidos, escalables y eficientes. Esto impulsa un futuro donde la inteligencia artificial distribuida sea la base para sistemas autónomos y colaborativos en una variedad de áreas.

REFERENCIAS

"FIPA ACL Message Structure Specification". Welcome to the Foundation for Intelligent Physical Agents. Accedido el 12 de noviembre de 2024. [En línea]. Disponible: http://www.fipa.org/specs/fipa00061/SC00061G.html

"FIPA Communicative Act Library Specification". Welcome to the Foundation for Intelligent Physical Agents. Accedido el 12 de noviembre de 2024. [En línea]. Disponible: http://www.fipa.org/specs/fipa00037/SC00037J.html

Jade Site | Java Agent DEvelopment Framework. Accedido el 12 de noviembre de 2024. [En línea]. Disponible: https://jade.tilab.com/doc/programmersguide.pdf