Janeiro Studio

Araujo, Pedro Bernabé

GITIA

Universidad Tecnológica Nacional, Facultad Regional Tucumán

Rodríguez, Sebastián Alberto

GITIA

Universidad Tecnológica Nacional, Facultad Regional Tucumán

Resumen

Este artículo tiene como finalidad presentar la herramienta de desarrollo Janeiro Studio y sus principales características. Janeiro provee soporte para la modelización de sistemas multiagentes utilizando la metodología ASPECS. La misma, fué desarrollada utilizando la plataforma Eclipse RCP permitiendo así dos modalidades posibles, "standalone" o plugin. Además permite el diseño, la verificación de los modelos y la gestión de los elementos presentes en los diagramas. En la actual fase de desarrollo en la que se encuentra Janeiro, el mismo brinda soporte a la primera fase de desarrollo de la mencionada metodología. Al final del artículo se presenta como ejemplo el protocolo de interacción FIPA modelado en Janeiro.

Palabras Clave Sistemas Multiagentes, metamodelo, CRIO, JANEIRO, CASE.

1 Introducción

Durante las últimas décadas la tecnología orientada a agentes se ha convertido en una de las herramientas más importante para el modelado de sistemas complejos, distribuidos y abiertos. En este sentido los sistemas basado en agentes han demostrado exitosamente su potencial abordando una amplia variedad de problemas que van desde la robótica, resolución de problemas distribuidos, modelado y simulación, solo por mencionar algunas áreas [23, 24]. El creciente interés se debe, entre otros, al concepto de agente como sistema autónomo. Un agente es una entidad física o virtual con un alto grado de autonomía, independiente, capaz de cooperar, competir, comunicarse, actuar flexiblemente y ejercer control sobre su comportamiento dentro del marco de sus objetivos. Un sistema multiagente está compuesto por múltiples agentes inteligentes que interactúan entre sí para alcanzar objetivos que pueden ser compartidos o no entre los agentes [22].

En la Ingeniería de Software Orientada a Agentes (ISOA o AOSE por sus siglas en inglés), requiere para el análisis, diseño e implementación de cuatro elementos fundamentales: el metamodelo y los lenguajes que se utilizarán para describir los modelos; la metodología que define la secuencia de pasos a seguir y los actores involucrados para la obtención de un diseño del producto; la plataforma de implementación sobre la cual se ejecutarán estos modelos; y por último la herramienta CASE (Computer Aided Software Engineering) utilizada para asistir al diseñador en el proceso de desarrollo. En este artículo presentamos los avances realizados en la herramienta CASE denominada Janeiro Studio para el metamodelo CRIO [17] y la metodología ASPECS [3]. La elección de la metodología ASPECS se debe a que soporta los conceptos de la teoría organizacional además de conceptos de la teoría de holones no disponibles en otras metodologías.

El resto de este artículo está organizado de la siguiente manera: Sección 2 se realiza un análisis de las herramientas que consideramos más relevantes en la literatura. Sección 3 se presenta los avances realizados en la herramienta Janeiro Studio. Sección 4 presentación

de como modelizar uno de los protocolos de interacción utilizando los diferentes diagramas disponibles. Y por último, en la sección 5 se detallan las conclusiones obtenidas y cual será el trabajo futuro para Janeiro Studio.

2 Trabajos relacionados

Una CASE es un conjunto de herramientas informáticas que asisten al diseñador en algunas de las actividades relacionadas con el desarrollo de un sistema (requerimientos, análisis, diseño, codificación y pruebas). Estas herramientas permiten transmitir lo más rápido posible las intenciones de los desarrolladores mediante una combinación de notaciones gráficas y textuales que representan algún lenguaje específico compartido por el equipo de desarrollo. Además, incrementan la productividad de los equipos de desarrollo reduciendo tiempo y costos a su vez que mejora la calidad del software entregado [18].

Existen diferentes herramientas presentes en la literatura para el modelado de sistemas multiagentes. En este trabajo nos enfocamos en aquellas que se encuentran centradas en los conceptos organizacionales (Organization-Centred MultiAgent System). Muchas de estas herramientas son para metamodelos específicos y brindan soporte visual para algunos o la mayoría de sus diagramas más relevantes.

Dentro de la categoría antes mencionada podemos destacar a agentTool III [7]. Es un entorno de desarrollo que asiste al usuario en el análisis, diseño e implementación de sistemas multiagentes basados en la metodología O-MASE [8]. Entre los modelos a los cuales da soporte están “Goal model”, “Agent Model”, “Role Model”, “Organizational Model”, “Protocol Model”, entre otros. Además la herramienta posee dos características adicionales: (i) módulo para validar la consistencia de los modelos planteados como así también la validación entre modelos; y (ii) un módulo específico que permite la generación de código para diferentes plataformas.

GAIA4E [2] es el nombre que recibe la herramienta creada para la metodología GAIA [25]. Cubre todas las fases de GAIA permitiendo al diseñador documentar los modelos correspondientes de acuerdo a una notación específica. Implementado como plugin para Eclipse Environment define su perspectiva conformada por tres vistas principales: Eclipse Navigator, GAIA Properties y Image Viewer. Además, la herramienta ofrece la posibilidad de cambiarse a cualquiera de las tres fases definidas en el proceso permitiendo que el usuario elija donde quiere trabajar.

Ingenias [16] también cuenta con una herramienta para el modelado de sistemas multiagentes llamado Ingenias Development Kit (IDK) [9]. Es un entorno visual rústico que permite la gestión del proyecto con múltiples diagramas, todos ellos organizados en paquetes, de acuerdo con la metodología. También, provee la posibilidad de crear repositorios de elementos que facilita su reutilización.

Si bien Prometheus [15] es una metodología considerada del tipo Agent-Centred MultiAgent System (ACMAS). La herramienta construida para la misma es interesante de mencionar. Esta se denomina Prometheus Design Tool (PDT) [20] y provee al diseñador del sistema de una interfaz gráfica intuitiva que facilita el desarrollo de varios de los artefactos definidos para la metodología. Adicionalmente, permite verificar la consistencia de los modelos propuestos realizando validaciones cruzadas entre varios modelos, propagación automática de los elementos de diseño cuando es posible y apropiado, etc.

Janeiro Studio surge de la necesidad de completar un ecosistema que ya cuenta con la metodología ASPECS y su respectivo metamodelo, además de la plataforma de ejecución Janus [6]. El desarrollo de este ecosistema a su vez, se debe a que no existe en la literatura una herramienta que brinde soporte a las características centrales de la teoría organizacional holónica como son el concepto de Capacidad y Holón que serán explicados más adelante.

3 Janeiro Studio

En los ’80 los Sistemas Multiagentes emergieron como una de las tecnologías más interesantes para abordar sistemas complejos. Entre ellos, el enfoque organizacional ha ganado especial importancia. Este está inspirado en la metáfora social y es usada tanto en metodologías (GAIA [26], MESSAGE [12] ASPECS) como metamodelos (AGR [5], MOCA [4], CRIO). Conceptos como “Organización”, “Grupo”,“Comunidad”, “Roles”, “Protocolos”

son términos comunes en este tipo de enfoque. En el presente artículo nos centraremos en el metamodelo CRIO.

3.1 Metamodelo CRIO

CRIO [17] es un metamodelo basado en los conceptos organizacionales. Su nombre resulta del acrónimo formado por sus cuatros conceptos principales:

Una *Capacidad* es la descripción de un know-how/servicio. En otras palabras es la especificación de una transformación de una parte de un sistema o de su ambiente. Es una abstracción de alto nivel que promueve la reusabilidad y modularidad y en este sentido puede ser considerado como un componente básico de diseño. Además, el concepto de capacidad permite la definición de un rol sin hacer ninguna suposición de la arquitectura interna de éste.

Un *Role* es definido como un comportamiento esperado (un conjunto de tareas del rol ordenados por un plan) y un conjunto de derechos y obligaciones dentro del contexto de la organización. El objetivo de cada rol es contribuir en alcanzar los requerimientos de la organización dentro del cual está definido.

Una *Interacción* es una secuencia de eventos dinámica intercambiada entre roles (una especificación de alguna ocurrencia que puede potencialmente disparar efectos en el sistema) o entre roles y entidades fuera del sistema.

Por último, una *Organización* se define como una colección de roles y sus interacciones dentro de un determinado contexto.

CRIO es la base fundamental de la metodología ASPECS. La mencionada metodología define un proceso de desarrollo de software. Establece los pasos a seguir cubriendo áreas claves que van desde los requerimientos hasta la codificación permitiendo un modelado de sistemas con diferentes niveles de abstracción. ASPECS está inspirado en el enfoque Arquitectura Dirigida por Modelos (o mejor conocida por sus siglas en inglés como MDA; Model Driven Architecture) que define tres niveles de modelos, cada uno de estos hace referencia a un metamodelo diferente. En la figura 1 se muestra la correspondencia entre los dominios de ASPECS y MDA.

En el nivel superior, y equivalente al Modelo Independiente de la Computación (CIM) de MDA, está presente el "Dominio del Problema" que provee una descripción organizacional del problema independiente de una solución específica,

El análogo al Modelo Independiente de la Plataforma(PIM) es el "Dominio de Agencia" que incluye los conceptos relacionados con una solución orientada a agentes para el problema analizado en la etapa anterior.

Por último, en el nivel inferior de ASPECS se encuentra el "Dominio de la Solución" que está relacionado con la implementación de la solución sobre una plataforma específica por lo que este dominio es dependiente de una plataforma de implementación particular. Este dominio se corresponde con el Modelo Específico de la Plataforma(PSM) de MDA.

En la teoría de sistemas multiagentes se pueden encontrar agentes que pueden a su vez estar compuestos de otros agentes. Estos sistemas jerárquicos de composición han recibido diferentes nombres tales como meta-agentes [10], grupos agentificados [14], entre otros. Entre ellos se encuentran los SMA Holónicos.

Un holón, por definición, es una estructura auto-similar compuesta de holones como subestructura. Un holon puede ser visto dependiendo del nivel de observación, como una entidad atómica o como una organización de holones [11]. Usando la perspectiva holónica, el diseñador puede modelar un sistema con entidades de diferente nivel de granularidad. Esta técnica recursiva permite generar modelos de subcomponentes desde un sistema complejo hasta alcanzar una etapa donde las tareas requeridas son manejadas por entidades atómica fáciles de implementar. El concepto de capacidad y la teoría de holones permite un manejo ágil y flexible de la tecnología agentes. Estos conceptos están implementados en ASPECS, CRIO y Janus, y a nuestro conocimiento no es posible encontrarlos en las otras metodologías, metamodelos o plataformas.

3.2 Descripción General

Janeiro Studio es un herramienta CASE multiplataforma de libre distribución, open-source, completamente desarrollada en JAVA que facilita el modelado de sistemas multiagentes basado en el enfoque organizacional. El objetivo de Janeiro es proveer un soporte adecuado tanto para el metamodelo organizacional CRIO como la metodología ASPECS.

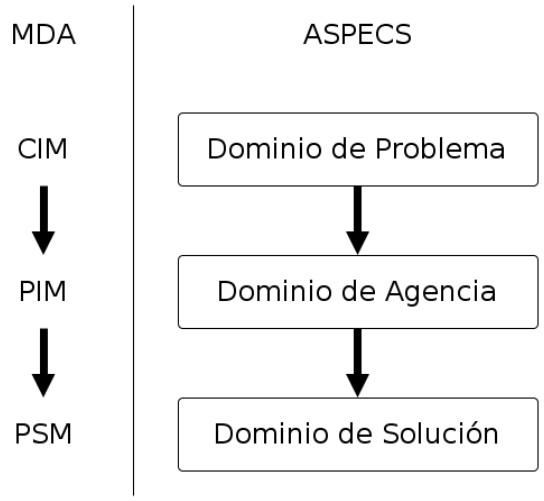

Figura 1: Dominios de ASPECS.

Para Janeiro Studio se utilizó Eclipse Rich Client Platform (RCP) [13] como base dada las ventajas que ofrece dicha plataforma para el desarrollo de herramientas integradas. Provee un entorno rico en opciones y debido a su popularidad posee una comunidad importante que le da soporte. También destacamos su desarrollo basado en el concepto de plugins que permite una alta modularización del sistema. Un plugin es definido como una unidad de modularidad en Eclipse, de hecho todo en Eclipse es desarrollado bajo este concepto. Básicamente, un plugin es autodescriptivo y explicita una lista de los otros plugins a los cuales depende para un funcionamiento adecuado.

El RCP nos permite una manera clara y sencilla de gestionar los plugins para construir aplicaciones complejas y funcionales. El metamodelo CRIO fue definido y adaptado en Eclipse Modeling Framework (EMF de ahora en mas) [19]. EMF es un framework creado por Eclipse Foundations que define una serie de actividades a seguir facilitando la construcción de herramientas personalizadas mediante la definición de un metamodelo de datos estructurados. Conjuga tres elementos importantes como JAVA, XML y UML [21].

Otro de los framework involucrados es Graphical Modeling Framework (GMF) útil para establecer como los conceptos serán representados gráficamente en el entorno de desarrollo propuesto.

Si bien Janeiro se encuentra en sus primeras etapas de desarrollo, el mismo provee soporte para cinco diagramas del metamodelo CRIO. Estos diagramas están representados mediante una notación gráfica similar a la utilizada en UML [21]. Los Diagramas de Requerimientos, Diagrama de Ontología, Diagrama Organizacional, Diagrama de Interacción y el Diagrama de Statechart; son los necesarios para cubrir la fase del "Dominio del Problema" definida en ASPECS (figura 1). Además, la herramienta, permite generar un esqueleto de código JAVA para la plataforma de Janus.

En la figura 2 se muestra el modelo equivalente de CRIO en EMF. Este modelo, denominado ecore, describe el metamodelo CRIO que posteriormente será manipulado a través de diferentes diagramas. Así los diagramas pueden considerarse "vistas" sobre el mismo modelo. EMF permite la creación dinámica de modelos definidos según un ecore específico. De esta forma, una vez definido el metamodelo CRIO, los diferentes diagramas son utilizados, según la metodología ASPECS, para definir el modelo final del sistema multiagentes. EMF fue seleccionado para este diseño debido a la estrecha integración con el ecosistema de modelización provisto por el proyecto Eclipse.

En la figura 3 se puede apreciar una captura de pantalla de Janeiro Studio. Para el mismo se definió una perspectiva particular que consta de cuatro vistas a saber:

Vista del Navegador(A) que indica en forma de árbol los elementos definidos por el usuario que están presentes en los diagramas, ésta estructura facilita la navegación y búsqueda de los conceptos.

Para representar el sistema la Vista del Editor(B) es donde se podrán manipular todos los diagramas que representan las distintas vistas y aspectos del sistemas.

Para cada uno de los tipos de diagramas se despliega una Vista de la Paleta(D) de los conceptos gráficamente representados que pueden utilizarse para la construcción del diagrama. Existe una paleta personalizada para cada editor.

Para los elementos gráficos vertidos en la Vista del Editor, el desarrollador puede especificar a través de la Vista de Propiedades ciertas propiedades de los conceptos que pueden figuran o no en la notación gráfica.

Por último, Janeiro Studio cuenta con una conjunto de reglas de validación que se encuentran implícitas en el metamodelo (por

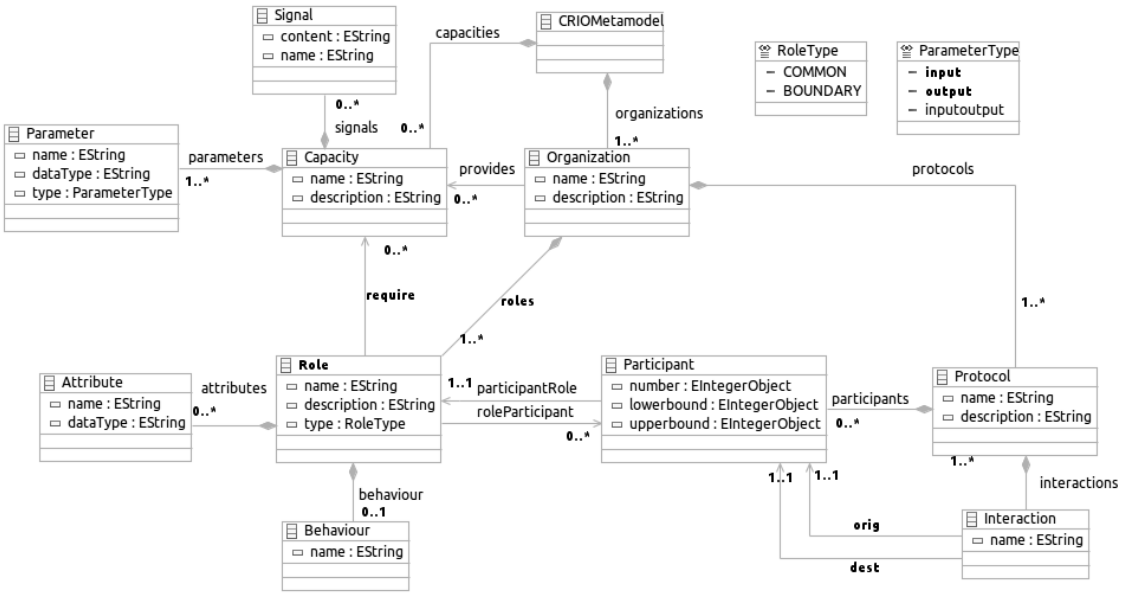

Figura 2: Metamodelo CRIO en EMF.

ejemplo: relaciones permitidas o cardinalidades) y otro conjunto de reglas definidas por el equipo de desarrollo que aseguran que los diagramas sean conceptualmente válidos (Por ejemplo: es obligatorio que los roles, organizaciones, capacidades y protocolos se deba especificar un nombre). Los resultados arrojados, producto de la aplicación de estas reglas en los modelos, son mostrados en la vista "Problems".

A continuación se describirán los diferentes diagramas soportados. Para más detalle de las fases e instancias del proceso en los que están involucrados, el lector puede referirse a la metodología ASPECS [3].

3.3 Diagrama de Descripción de Requerimientos del Dominio

Para que el desarrollo de un sistema sea exitoso, el mismo debe contar con una correcta documentación de sus requerimientos tanto funcionales como no funcionales. En otras palabras, el objetivos principal de la fase de requerimientos es capturar las necesidades o intereses de las personas que serán afectadas por el desarrollo del sistemas además de contar con una descripción global del comportamiento de la aplicación objeto de desarrollo. La metodología de ASPECS define para tal objetivo el diagrama "Descripción de los Requerimientos del Dominio" (figura 4) inspirado en el Diagrama de Casos de Usos de UML permitiendo representar las descripciones funcionales y no funcionales utilizando un lenguaje específico del dominio provisto por el usuario.

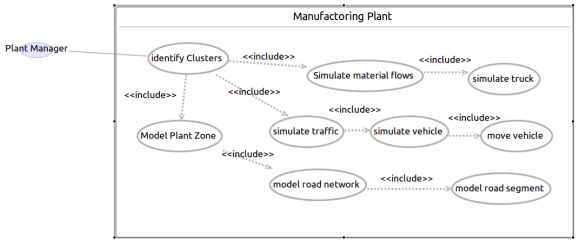

Figura 4: Diagrama de Descripción de Requirimientos del Dominio. Fragmento extraido del artículo [3].

3.4 Diagrama de Ontología del Problema

Los requerimientos de los usuarios en su mayoría son expresados utilizando el lenguaje natural. A su vez, estos requerimientos tienen implícitos términos propios del dominio de aplicación que son extraídos de los casos de usos definidos en el Diagrama de Requerimientos del Dominio. En el Diagrama de Ontología del Dominio (DOD)(figura 5) permite

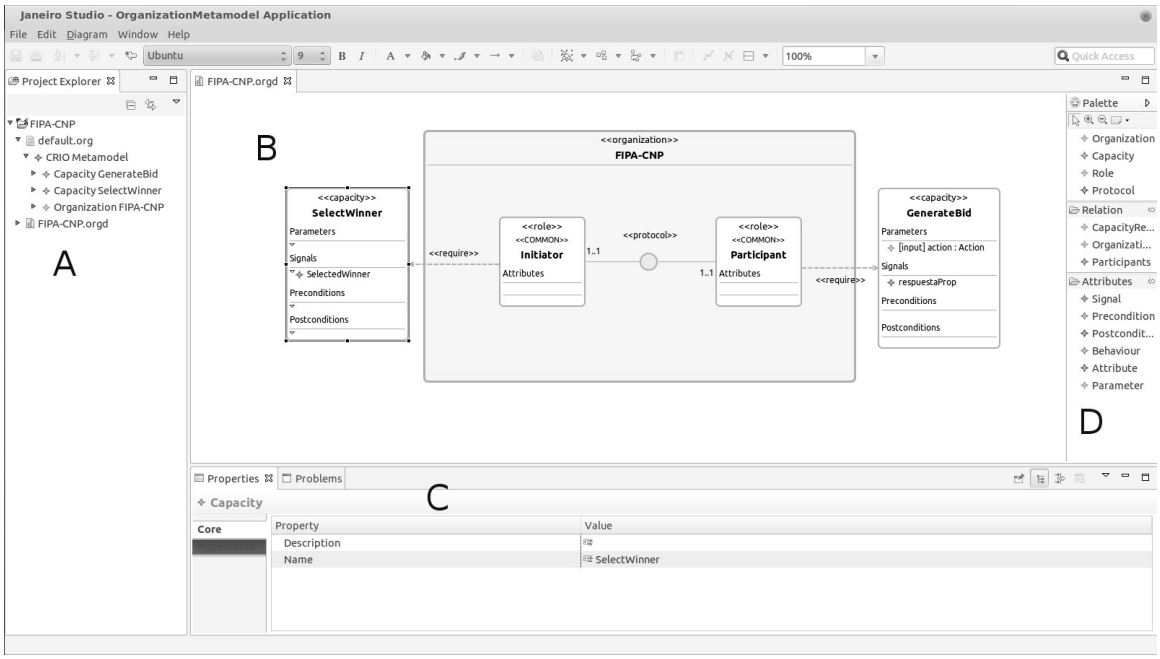

Figura 3: Captura de pantalla de Janeiro Studio

identificar, modelados como un diagrama de clases, los conceptos, predicados y acciones relevantes para el dominio del problema.

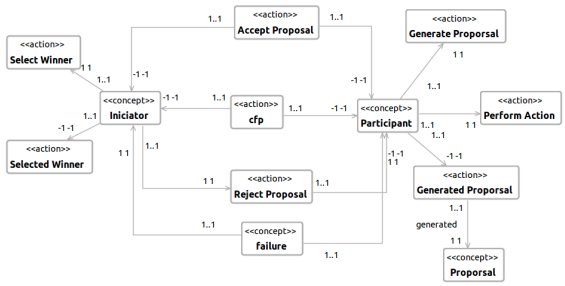

Figura 5: Diagrama de Ontología del Dominio.

3.5 Diagrama Organizacional

El principal diagrama del metamodelo CRIO es el "Diagrama Organizacional"(figura 6), el mismo está compuesto de organizaciones, roles, protocolos y capacidades. Una organización está representada mediante una caja rectangular con el estereotipo de "≪organization≫" e inmediatamente más abajo posee una etiqueta con el nombre de la organización. El mismo contiene todos los roles que estén involucrados en la organización, los protocolos y las relaciones entre los roles y protocolos.

De igual manera que las organizaciones, el concepto de rol está representado por un caja con el estereotipo de "≪role≫" junto con otro que define el tipo de rol que queremos modelar. Así se encuentran: rol "≪common≫" ubicado dentro del sistema y capaz de interactuar con otros roles del mismo tipo además del rol Boundery. El rol "≪boundery≫" representa aquellos elementos que se encuentran ubicados en los límites o frontera entre el sistema y el exterior y es responsable de las interacciones que ocurren en ese límite. Inmediatamente más abajo de estas etiquetas podemos colocar el nombre que deseamos para el rol. Con respecto a su cuerpo, en el mismo existen dos secciones; la primera relacionada con los atributos que serán utilizados para el comportamiento; y la segunda permite definir el Diagrama de Statechart asociado al rol.

Mediante el concepto de protocolo es posible descomponer los objetivos organizacionales. En otras palabras, un protocolo de interacción y los roles asociados al mismo indican cuales son los roles y la secuencia de mensajes necesarios para alcanzar uno de los objetivos definidos para la organización. El protocolo en el diagrama está representado por un círculo con una etiqueta afuera del mismo que indi-

ca el estereotipo "≪protocol≫". La cardinalidad de ésta asociación representa el número de instancias de agentes que podrán jugar este rol por cada instancia de organización.

Una capacidad es la descripción de un know-how/servicio. En otras palabras es una especificación de una transformación de una parte del sistema o su ambiente. Una capacidad en Janeiro Studio, representada por una clase estereotipada con una etiqueta "≪capacity≫", se ubica fuera de la organización. Esto se debe a que el concepto de capacidad fue concebido como una abstracción de alto nivel que promueve la reusabilidad y modularidad y que puede proveer sus características a diferentes organizaciones. El concepto de capacidad posee una serie de secciones que permiten definir cuestiones relacionadas a la misma:

- Parámetros: Estos son parámetros de entrada de la capacidad y está conformada por el nombre del parámetro y el tipo de dato que representa.
- Señales: Los servicios que implementan una capacidad disparan señales dirigidas al rol para indicar la finalización de la actividad. Y, potencialmente parámetros para comunicar los resulados de la ejecución.
- Pre- y pos- condiciones: Son restricciones lógicas definidas tanto para los valores de entradas como los de salida, respectivamente.

Cuando una capacidad es requerida por un rol, esta se debe asociar con un enlace dirigido. Esta relación posee un rótulo denominado "≪require≫". Estas relaciones indican cuales son las competencias necesarias que deberá tener el agente para poder tomar ese rol.

3.6 Diagrama de Interacción

La finalidad del diagrama de interacción (figura 7) es describir la secuencia de paso de mensajes entre roles que tiene lugar en un protocolo. El diagrama, que describe a un protocolo del diagrama organizacional, está compuesto por representantes, mensajes o interacciones, llamados a capacidades y señales.

Gráficamente un participante consta de un "RoleHead" que es un rectángulo con una etiqueta "≪participant≫". Además, posee una línea vertical punteada que representa la línea de vida del rol que se corresponde a su función "live" y es donde se originan o es receptor de los mensajes como así también de los llamados a capacidad o señales. En el "RoleHead" también es posible establecer el nombre de instancia además del nombre del rol; utilizando el formato "nombreInstancia:NombreRol". Donde "nombreInstancia" se define el/los destinatario(s) o receptor(es) de los mensajes que llegan al participante. Tres tipos de mensajes son posible en el diagrama: Si el mensaje es de tipo broadcast, denotado por "*" como nombre de instancia, el mensaje tiene como destinatario a todos los agentes que juegan el rol; en cambio, si el mensaje está dirigido a un destinatario elegido al azar por la plataforma se utilizará el símbolo "?"; por último, si el mensaje está dirigido a un roleplayer en particular basta poner la dirección del agente ("AgentAddress") o nombre.

La interacción o mensajes están representados por un línea dirigida con una etiqueta indicativa del nombre de la interaccion además de los parámetros asociados a dicho mensaje.

Los roles tienen asociados tareas u operaciones, además de los llamados a capacidades, que representan su comportamiento esperado dentro de la organización. Dentro de estas operaciones existen dos tipos especiales denominados "call" y "onSignal" que especifica un llamado a capacidad o una señal que levanta un servicio indicando la finalización de una actividad.

3.7 Diagrama Statechart

En el rol del diagrama organizacional se puede observa una sección referente a su comportamiento, denominado ≪behavior≫. Los estados de un rol se refieren al conjunto de valores de sus atributos. Este comportamiento está representado por un diagrama de Statechart (figura 8). El diagrama está compuesto básicamente por estados y transiciones entre los estados. En la paleta del diagrama de se pueden encontrar tres tipos de estados: los pseudo estados Inicial, Final y los estados normales. Estos estados se pueden encontrar en la paleta del diagrama; el estado inicial, representado por un circulo negro, el estado final por un

circulo dentro de otro también de color negro y los estados normales como una caja rectangular.

Las transiciones son las relaciones entre los estados y están representadas gráficamente por flechas dirigidas con una etiqueta. La etiqueta mencionada consta de tres partes “evento/condición/acción".

4 Ejemplo

En esta sección presentaremos uno de los protocolos más aceptado por la comunidad científica denominado FIPA-Contract Net Protocol [1] modelizado en Janeiro Studio.

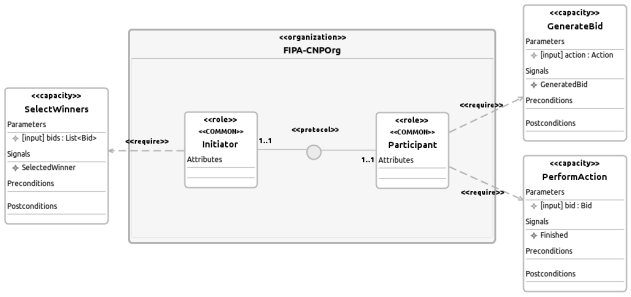

Figura 6: Diagrama Organizacional.

Los dos elementos principales del protocolo FIPA son “Initiator”(Iniciador) y “Participant”(Participante), estos son modelados como roles dentro de la organización denominada “FIPA-CNPOrg”; a su vez, estos roles interactúan entre sí a través de los protocolos que definen una secuencia ordenada de interacción describiendo el paso de mensajes existentes entre los participantes (en un protocolo los roles toman el nombre de participantes). Además, tanto iniciador como participante tienen asociados capacidades que se explicarán a medida que se desarrolle el ejemplo.

El funcionamiento FIPA-CNP empieza cuando el Iniciador envía a todos los participantes un acta de propuestas a través de un mensaje denominado “Call for proporsal”. Lo anterior, es un mensaje del tipo broadcast y está diagramado en el Diagrama de Interacción mediante una flecha con el rótulo “cfp(action:Action)” donde cfp es el nombre del mensaje y ”action:Action“ es el parámetro que el roleplayer del Iniciador está tratando de transmitir a los participantes; para esta situación el nombre del participante tiene el formato “*:Participant”. Los participantes reciben el mensaje cfp y cada uno, independientemente, evalúa la propuesta mediante la invocación de la operación especial “call”, en nuestro ejemplo “call(GenerateProposal, action)” dónde el primer parámetro es la capacidad requerida y el resto, si los hubiera, son los parámetros de entrada de la capacidad.

Una vez finalizada la tarea de evaluación se dispara un estímulo al rol, denominado “signal”, indicando la aceptación, rechazo o simplemente que no se entendió el mensaje. El formato para signal es “signal(GeneratedProposal,proposal:Proposal)”, y al igual que el llamado a capacidad, el primer parámetro es el nombre de una de las señales definida en la sección Signals de la capacidad y el resto son los valores de retorno de la capacidad.

Para evitar que Iniciador espere indefinidamente las respuestas de los participantes, se establece un límite de tiempo para la recepción de mensajes. Una vez que expira dicho tiempo realiza un llamado a la capacidad “SelectWinner” evaluando una lista de propuestas(proposal) de sólo aquellos que aceptaron y determina, bajo ciertos criterios establecidos, que participante o participantes (también puede no elegir alguno) son los que llevarán a cabo las tareas/acciones.

Él o los participantes que resultaron seleccionados recibirán del iniciador un mensaje de aceptación (accept-proporsal()) para que lleve a cabo la ejecución de las tareas y aquellos que no fueron seleccionados un mensaje de rechazo(reject-proposal()) al subconjunto de participantes rotulado como “bidderswinner:Participant". Por último, y una vez finalizadas las tareas que el o los participantes se comprometieron a realizar, se puede dar dos tipos de situaciones: las tareas fueron realizadas (se puede también enviar un mensaje más explicativo del tipo “inform-result”) o indicando que no pudo realizar la tarea mediando un mensaje del tipo “failure()”.

En la figura 8 se muestra los posibles estados en el cual puede estar el rol participante.

5 Conclusiones y trabajos futuros

Este artículo presenta los avances realizados en Janeiro Studio. El mismo brinda soporte a

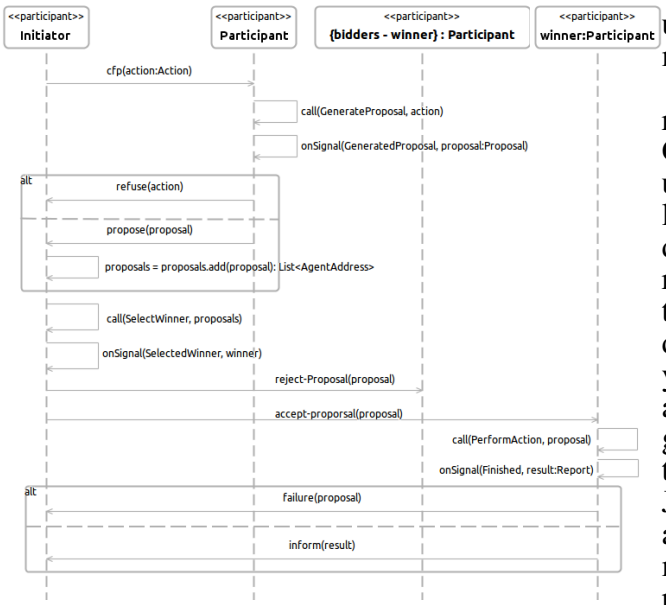

Figura 7: Diagrama de Interacción.

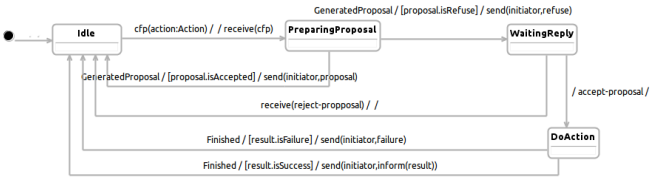

Figura 8: Diagrama de Interacción.

la primera fase de la metodología ASPECS. Diagrama de Requerimientos del Dominio útil para capturar las necesidades de los usuarios. El Diagrama de Ontología describe los términos usados en el lenguaje específico del dominio de aplicación. El Diagrama Organizacional en donde se define las organizaciones y dentro de los mismo la colección de roles, protocolos e interacciones. El Diagrama de Interacción, donde se representa el intercambio de mensajes y la secuencia asociados a los mismos. Por último, el Diagrama de Statechart que permite representar los estados posibles en los que puede estar un rol.

El próximo paso está previsto realizar los diagramas para el dominio de agencia. Los mismo requiere la definición de un conjunto de reglas de transformación que nos permita generar los elementos necesarios a partir de los modelos creados para el "dominio del problema". A su vez, se encuentra en desarrollo un módulo que permitirá verificar y validar los modelos diseñados.

Además, está previsto modelizar con Janeiro Studio el funcionamiento de un "Smart-Grid" para simular el consumo de energía de una población determinada (en nuestro caso, la provincia de Tucumán, Argentina) y así poder determinar cuales son las acciones necesarias para acompañar la creciente demanda tanto en la generación como en la transmisión y distribución de la energía eléctrica. Este proyecto permitiría, como uno de sus principales aspectos, el uso racional y eficiente de la energía a su vez que se reduce el impacto ambiental. Este trabajo permitirá validar la utilidad de Janeiro Studio y refinar la herramienta como así también los conceptos del metamodelo y la metodología adoptadas para el desarrollo del proyecto.

Como hemos mencionado anteriormente, nuestra intención con Janiero Studio es brindar al diseñador de sistemas de una herramienta que provea el mayor soporte posible a la metodología ASPECS. Además permitir la generación de código, procesos de debbuging e integración con la plataforma Janus. Logrando así contribuir en el armado de un ecosistema para abordar sistemas complejos mediante la utilización de modelos basados en sistemas multiagentes holónicos.

Referencias

[1] FIPA Contract Net Interaction Protocol Specification. Technical Report SC00029H, Foundation for Intelligent Physical Agents (FIPA), 2002.

[2] Luca Cernuzzi and Franco Zambonelli. Gaia4e: A tool supporting the design of mas using gaia.

[3] Massimo Cossentino, Nicolas Gaud, Vincent Hilaire, Stéphane Galland, and Abder Koukam. Aspecs: an agent-oriented software process for engineering complex systems. *Autonomous Agents and Multi-Agent Systems*, 20(2):260 – 304, 2010.

[4] J. Ferber and O. Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems. In *Proceedings of the 3rd International*

Conference on Multi Agent Systems, ICMAS '98, pages 128–, Washington, DC, USA, 1998. IEEE Computer Society.

[5] Jacques Ferber, Olivier Gutknecht, and Fabien Michel. From agents to organizations: An organizational view of multiagent systems. In Paolo Giorgini, JörgP. Müller, and James Odell, editors, *Agent-Oriented Software Engineering IV*, volume 2935 of *Lecture Notes in Computer Science*, pages 214–230. Springer Berlin Heidelberg, 2004.

[6] Stéphane Galland, Nicolas Gaud, Sebastian Rodriguez, and Vincent Hilaire. Janus: Another yet general-purpose multiagent platform. In Massimo Cossentino, R.F. Fernandez, and F. Migeon, editors, *7th Agent-Oriented Software Engineering Technical Forum (TFGAOSE-10)*. Agent Technical Fora, 2010.

[7] Juan C. Garcia-Ojeda, Scott A. DeLoach, and Robby. agentTool III: from process definition to code generation. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '09, pages 1393–1394, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems.

[8] JuanC. Garcia-Ojeda, ScottA. DeLoach, Robby, WalamitienH. Oyenan, and Jorge Valenzuela. O-mase: A customizable approach to developing multiagent development processes. In Michael Luck and Lin Padgham, editors, *Agent-Oriented Software Engineering VIII*, volume 4951 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin Heidelberg, 2008.

[9] Jorge J. Gomez-Sanz, Rubén Fuentes, Juan Pavón, and Ivan García-Magariño. Ingenias development kit: a visual multiagent system development environment. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: demo papers*, AAMAS '08, pages 1675–1676, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

[10] John H. Holland. *Hidden order: how adaptation builds complexity*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1995.

[11] Arthur Koestler. *The Ghost in the Machine*. Hutchinson, 1967.

[12] Francisco Leal and João Rodriguez. Message: Methodology for Engineering Systems of Software Agents. Technical report, Telecom Italia Lab and PT Inova ccãl, 2001.

[13] Jeff McAffer, Jean-Michel Lemieux, and Chris Aniszczyk. *Eclipse Rich Client Platform*. The Eclipse Series. Pearson Education, 2 edition, May 2010.

[14] James Odell, Marian Nodine, and Renato Levy. A metamodel for agents, roles, and groups. pages 78–92. Springer, 2005.

[15] Lin Padgham and Michael Winikoff. Prometheus: A methodology for developing intelligent agents. In Fausto Giunchiglia, James Odell, and Gerhard Weiß, editors, *Agent-Oriented Software Engineering III*, volume 2585 of *Lecture Notes in Computer Science*, pages 174–185. Springer Berlin Heidelberg, 2003.

[16] Juan Pavón and Jorge Gómez-Sanz. Agent oriented software engineering with ingenias. In Vladimír Mařík, Michal Pěchouček, and Jörg Müller, editors, *Multi-Agent Systems and Applications III*, volume 2691 of *Lecture Notes in Computer Science*, pages 394–403. Springer Berlin Heidelberg, 2003.

[17] Sebastian Rodriguez, Nicolas Gaud, Vincent Hilaire, Stéphane Galland, and Abder Koukam. An analysis and design concept for self-organization in holonic multi-agent systems. In *Proceedings of the 4th international conference on Engineering self-organising systems*, ESOA'06, pages 15–27, Berlin, Heidelberg, 2007. Springer-Verlag.

[18] Ian Sommerville. *Software Engineering*. Addison-Wesley, 9 edition, March 2010.

[19] David Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks.

EMF: Eclipse Modeling Framework 2.0. Addison-Wesley Professional, 2nd edition, 2009.

[20] John Thangarajah, Lin Padgham, and Michael Winikoff. Prometheus design tool. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, AAMAS '05, pages 127–128, New York, NY, USA, 2005. ACM.

[21] Unified Modeling Language (UML) Superstructure V 2.4.1, August 2011. Version 2.4.1.

[22] Gerhard Weiss. *Multiagent Systems*. Intelligent Robotics and Autonomous Agents. MIT Press, Second edition, March 2013.

[23] Michael Wooldridge. *An Introduction to MultiAgent Systems*. Wiley Press, 2009.

[24] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2), 1995.

[25] Michael Wooldridge, NicholasR. Jennings, and David Kinny. The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

[26] Franco Zambonelli, Nicholas R. Jennings, and Michael Wooldridge. Developing multiagent systems: the gaia methodology. *ACM Transactions on Software Engineering and Methodology*, 12(3), July 2003.