We see some potential in reusing code-analysis techniques in order to perform end reachability checks for story telling. Another research line we consider interesting for the future is trying to develop objective metrics for estimating some of the psychological effects achieved by a story in the player. The task seem not easy, but such a metric would be useful while trying to explain why a story may be more engaging than other.

References

1. Propp, V.: Morphology of the Folktale. 2nd edn. University of Texas Press (June 1968)
2. Zurawsky, R., Zhou, M.: Petri nets and industrial applications: A tutorial. IEEE Transactions on Industrial Electronics **41**(6) (December 1994) 567–583
3. Natkin, S., Vega, L.: A petri net model for computer games analysis. Int. J. Intell. Games & Simulation **3**(1) (2004) 37–44
4. Papert, S.: Mindstorms: Children, Computers, and Powerful Ideas. Basic Books, New York (1980)
5. Collé, F., Champagnat, R., Prigent, A.: Scenario analysis based on linear logic. In: ACE '05: Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology, New York, NY, USA, ACM Press (2005) 1
6. Prigent, A., Chanpagnat, R., Estreillier, P.: Driving stories, benefits of properties analysis. In: 7th International Conference on Computer Games. (2005)
7. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Adisson Wesley (1994)
8. Johnson, S.: Lint, a C program checker. Technical report, Bell Laboratories (1977)
9. McCabe, T.J.: A complexity measure. IEEE Transactions on software engineering **SE-2**(4) (December 1974)
10. Kincaid, J.P., Fishburne, R.P.J., Rogers, R.L., Chissom, B.S.: Derivation of new readability formulas (automated readability index, fog count and flesch reading ease formula) for navy enlisted personnel. Technical report, Naval Technical Training, U. S. Naval Air Station, Memphis (1975)
11. Shoham, Y.: Agent-oriented programming. Artif. Intell. **60**(1) (1993) 51–92
12. Ierusalimschy, R., de Figueiredo, L.H., Filho, W.C.: Lua — an extensible extension language. Software: Practice and Experience **26**(6) (1996) 635–652
13. White, J.E. In: Mobile agents. MIT Press, Cambridge, MA, USA (1997) 437–472
14. Sunshine-Hill, B.: Pluto 2.0. Available at http://luaforge.net/projects/pluto/ (2005)

Modelando el Proceso de Desarrollo de INGENIAS con EMF[1]

Iván García-Magariño, Alma Gómez-Rodríguez and Juan C. González

[1]Dpto. de Ingeniería del Software e Inteligencia Artificial.
Universidad Complutense de Madrid (SPAIN)
ivan_gmg@fdi.ucm.es - http://grasia.fdi.ucm.es/gschool

[2] Dpto. de Informática. Universidad de Vigo.
Ed. Politécnico, Campus As Lagoas, 32004 Ourense
alma,jcmoreno@uvigo.es - http://gwai.ei.uvigo.es

Abstract. Desde que el "FIPA Methodology Technical Comité" se fijó como una de sus principales actividades la creación de un metamodelo que pueda ser usado para describir procesos de desarrollo válidos para la mayor parte de las metodologías de desarrollo de Sistemas Multi-Agente existentes; varios procesos de desarrollo de distintas metodologías han sido modelados utilizando SPEM v.1.2. En este artículo se presenta no sólo una propuesta de modelado para el proceso de INGENIAS, sino también un método para realizar la especificación de otros procesos y una herramienta de ayuda basada en el Eclipse Modeling Framework (EMF). El hecho de poseer una especificación de procesos de desarrollo para una metodología permite la identificación de partes(fragmentos) comunes a varias metodologías y también la propuesta de un "estándar de desarrollo", tal y como se propone en el anteriormente citado comité técnico de FIPA; pero, además, es de particular interés desde el punto de vista industrial para construir herramientas específicas de ayuda y control del desarrollo de aplicaciones, cuestión crítica en el caso de Sistemas Multi-Agente.

1. Introducción.

El paradigma de desarrollo orientado a Agentes ha demostrado su capacidad en el modelado y construcción de sistemas complejos, distribuidos y de gran tamaño. Por ello, ha constituido un campo importante de desarrollo e investigación en el ámbito del desarrollo de sistemas y particularmente de la Ingeniería del Software. En esta disciplina la garantía de calidad del sistema desarrollado se fundamenta en la definición de metodologías; esto, unido al gran auge de los Sistemas Multi-Agente (SMA), ha supuesto la definición de múltiples metodologías semiformales orientadas a Agentes [2], [5], [7], [12], [13], [17].

En el ámbito de garantía de calidad antes mencionado, una de las principales líneas de trabajo actuales es el estudio y mejora de los procesos llevados a cabo para el

[1] Investigación financiada por el proyecto nacional *Métodos y herramientas para modelado de sistemas multi-agente* del Ministerio de Educación y Ciencia TIN2005-08501-C03-01.

desarrollo y mantenimiento del software [6]. Esta línea de actuación se basa en la correlación existente entre calidad de proceso y calidad de producto obtenido. Por todo ello, es importante la obtención de modelos de los procesos de desarrollo empleados, considerando que un Modelo de Proceso de Software o SPM (Software Process Model) es una descripción de los aspectos estructurales y de comportamiento de un proceso en el ámbito del desarrollo de software, usando como formalismo algún lenguaje de modelado de procesos o PML (Process Modeling Language) [3].

En los últimos años, el modelado de procesos de software basados en agentes está adquiriendo una importancia creciente como mecanismo que debe permitir, por un lado, una mejor comprensión de ese proceso con vistas a su evaluación y mejora y, por otro, la posibilidad de lograr un cierto grado de automatización del mismo, tal como es norma en otras disciplinas de la ingeniería. Un reto fundamental de la modelización de procesos de software es encontrar un PML estándar para la descripción de los mismos. Desde el punto de vista del modelado de procesos de desarrollo software basados en SMA, el *FIPA Methodology Technical Committee* ha sugerido la utilización del *Software Process Engineering Metamodel* (SPEM) propuesto por el *Object Management Group* (OMG) como un perfil de UML con el que modelar los procesos de desarrollo software. Este estándar se encuentra con una versión estable la 1.1 (http://www.omg.org/technology/documents/formal/spem.htm) y otra la 2.0 en periodo de revisión y que adecua la propuesta original al estándar de UML-2.1.1 (http://www.omg.org/technology/documents/formal/uml.htm). En este artículo se propone un método para realizar la especificación del proceso de desarrollo de INGENIAS [8], [17] con la versión estable SPEM v 1.1, pero aplicando algunas de las propuestas presentes en el nuevo estándar 2.0, de manera que el metamodelo resultante pueda ser actualizado sin cambios importantes a la nueva versión.

Un proceso de desarrollo software puede ser entendido, a un elevado nivel de abstracción, como un simple grafo de dependencias entre tres componentes fundamentales: los trabajadores o participantes en el proceso (*roles* o *workers*), los productos generados y consumidos (*work product*) y las actividades (*activities*) y tareas (*tasks*) que se emplean en ese proceso y que constituyen instancias particulares de los trabajos (*work definition*) a realizar. Sin embargo, cualquier proceso de desarrollo software es lo suficientemente complejo como para necesitar además guías o recomendaciones (*guidance*) sobre las actividades a realizar y el momento en que deben ser realizadas. A pesar de esta interpretación simplista de la idea de proceso de desarrollo, hay que indicar que el modelado de los procesos de desarrollo, al igual que ocurre con el desarrollo de aplicaciones software, presenta diversas vistas ortogonales y complementarias entre sí. Cada una de estas vistas ayuda a comprender todo el proceso en su conjunto y permite especificarlo de una manera progresiva.

Por otra parte, la especificación y la documentación necesarias para entender un proceso de desarrollo de software orientado a agentes no pueden ser desarrolladas en unas pocas páginas, por lo que en este artículo se ha decidido presentar (en la sección tercera) una propuesta de marco de desarrollo basada en tres vistas ortogonales: *Ciclo de vida, Disciplinas* y *Guías*, de las que presentamos fragmentos significativos de las dos primeras y damos una ligera indicación de la tercera vista que complementa y documenta la manera en que se debe gestionar e implantar el proceso de desarrollo especificado. Con objeto de comprender las especificaciones aportadas se ha decidido presentar previamente en la segunda sección el editor utilizado en la especificación y

que ha sido desarrollado sobre EMF siguiendo la especificación v1.1 de SPEM. En la sección cuarta se determinan las disciplinas de INGENIAS. Por último se presentan las conclusiones y trabajo futuro.

2. El Editor de SPEM con EMF: Construido a partir de su Metamodelo.

El problema de obtener una herramienta de edición de modelos de un lenguaje de modelado es un problema compartido con el área de los lenguajes específicos de dominio o *Domain-Specific Model Languages* (DSML) [1]. Por tanto, se adopta una solución similar a la utilizada en esta área. Se define un modelo del lenguaje (un metamodelo) y se utiliza un framework, que dado dicho metamodelo, genera el editor del lenguaje de modelado. De esta manera, el desarrollo de un editor para un lenguaje de modelado se reduce a la correcta definición del metamodelo, lo que limita el coste de producción del editor y facilita su mantenimiento.

En estos momentos hay varios lenguajes de metamodelado disponibles. Entre otros, se pueden citar el lenguaje usado por MOF [15], GOPRR [11], [18] y Ecore (usado por *Eclipse Modelling Framework* (EMF) [4], [14]). Por lo que respecta a este artículo, es importante indicar que la especificación de SPEM [16] de OMG está definida usando el lenguaje MOF, mientras de las herramientas de metamodelado, EMF es el framework más estable para la generación de editores. Por esto, se ha traducido SPEM al lenguaje Ecore (usado por EMF). De esta manera se ha generado automáticamente el editor presentado en este artículo. Los editores creados con EMF, producen modelos representados con documentos XMI, fáciles de analizar e importar.

2.1. Características Estructurales de SPEM.

En este artículo, se han extraído las características estructurales propias de SPEM que se han de tener en cuenta en la definición del metamodelo en Ecore, y que repercuten en los documentos XMI generados por el editor que representan procesos de desarrollo de software. Éstas características estructurales son las siguientes:

1. Entidades con atributos. En SPEM se tienen ciertas entidades con unas características o atributos, por ejemplo, *WorkDefinition* o *Process*.
2. Relaciones con atributos. Por ejemplo, la relación *Precedes* tiene el atributo enumerado *PrecedesKind*
3. Relaciones con atributos en los extremos. Por ejemplo, en SPEM el extremo de la relación *Association* tiene, entre otros, los atributos *isNavigable* y *multiplicity*.
4. Relaciones n-arias. Por ejemplo, la entidad *WorkDefinition* tiene una relación n a n llamada *ParentWork*, que relaciona un conjunto de padres con un conjunto de hijos.

2.2. Los Documentos XMI Generados por el Editor para Representar Procesos Software.

Teniendo en cuenta las características estructurales del apartado anterior, el editor de SPEM usa documentos XMI con la representación explicada a lo largo de este apartado. El editor generado automáticamente (Fig. 3) permite editar de manera visual documentos XMI con dicha representación.

Representación de las Entidades. Las entidades de SPEM se representan con elementos XML. Los atributos de estas entidades se representan con los atributos XML. Por ejemplo, se puede definir un ciclo de vida llamado INGENIAS, con el siguiente elemento XML:

```
<wdef xsi:type="plc_entities:Lifecycle" name="INGENIAS"/>
```

Representación de las Relaciones. Cada relación de SPEM se representa con un elemento XML para el cuerpo y un elemento XML para cada extremo. Por ejemplo, a continuación se define una relación *precedes*, con el atributo *pKind* con el valor *pk\fn\st*, y con los extremos *supplier* y *client*.

```
<dep xsi:type="dep_relations:Precedes" pKind="pk_fn_st">
  <supplier EDMod="//@plc/@plcent/@wdef.1"/>
  <client EDMod="//@plc/@plcent/@wdef.0"/>
</dep>
```

Cada extremo está conectado a otro elemento del modelo, indicándose la ruta donde se encuentra dicho elemento. Para la ruta se usa el lenguaje *XPath*, un ejemplo de ruta es *//@plc/@plcent/@wdef.1*. Con esta representación, se puede tener relaciones con atributos tanto en el cuerpo como en los extremos. Además se podrían tener tantos extremos como se quisieran, por lo que es posible disponer de relaciones n-arias.

Representación de la Especificación. El elemento raíz del documento XMI generado por el editor es *Specification*. Dentro del elemento raíz hay elementos para cada paquete original de SPEM(*Core, Actions, State\Machines, Activity\Graphs, Model\Management, BasicElements, Dependencies, Process Structure, Process Components* y *Process Lifecycle*). La etiqueta de estos elementos son las iniciales de los paquetes originales o abreviaturas. Algunos de ellos contienen a su vez un elemento para entidades y otro para relaciones.

3. Guía de modelado de procesos de desarrollo de software orientado a agentes.

A lo largo de esta sección se presenta mediante un caso práctico (proceso de desarrollo unificado de INGENIAS) un marco para la especificación y modelado de procesos de desarrollo software orientados a agentes. En la realización de la

especificación se ha empleado el editor introducido en la sección anterior. El marco propuesto se basa en el desarrollo de tres vistas ortogonales entre sí: la vista del ciclo de vida del proceso, la vista de las disciplinas empleadas en el proceso y la vista de guías y recomendaciones sobre los productos, trabajos y roles que participan en los elementos de modelado que describen las otras dos vistas. Esta última es una vista de documentación del proceso que, aunque no se detalla en los siguientes apartados, está presente en la redacción de los mismos. La primera de las vistas propuestas describe el recorrido que se debe cubrir para la obtención como producto final de una Sistema Multi-Agente y las actividades a realizar en dicho recorrido. El establecimiento de las disciplinas que participan en el proceso (segunda vista) delimita los roles y participantes en dicho proceso junto con las tareas que deben desempeñar y los productos que producirán y consumirán.

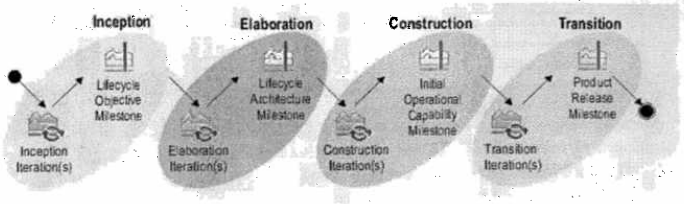

Fig. 1. Proceso de desarrollo unificado de INGENIAS

El caso de estudio escogido para presentar esta guía de modelado ha sido el *proceso de desarrollo unificado de INGENIAS* [8], [9]. El proceso se plantea como una integración de los metamodelos propuestos por la metodología INGENIAS con el "Unified Development Process" (UDP) [10] en lo referente a las disciplinas de análisis y diseño. En el UDP, las tareas a desarrollar en el análisis y el diseño de las aplicaciones se encuentra distribuido en tres fases consecutivas: *Inception, Elaboration*} y *Construction* que se estructuran en varias iteraciones (ciclos completos de desarrollo que suponen la realización de diversas tareas de análisis, diseño, implementación y pruebas) con las que se construye gradualmente el sistema. En la Fig.1 se presenta de manera gráfica el proceso de desarrollo sugerido.

3.1. El ciclo de vida de INGENIAS.

Para comenzar la especificación del proceso de desarrollo a aplicar, se sugiere empezar por modelar el ciclo de vida concreto que se quiere implantar, ya que permite abstraerse de los participantes y de los productos a conseguir y centrarnos en la especificación del recorrido y de las actividades a realizar en cada momento. En el caso de INGENIAS hay que dividir el recorrido en fases consecutivas y cada una de dicha fases en un número de iteraciones diferente en función de la fase en la que nos encontremos.

A la hora de especificar las iteraciones, uno de los inconvenientes de la versión 1.1 de SPEM es la ausencia de un lenguaje formal con el que poder indicar restricciones sobre el número de iteraciones admisibles. Para solucionar esta carencia, se sugiere

incorporar al nombre de la iteración el rango de iteraciones mínimo y máximo admisibles (por ejemplo [1..3]), siempre que el conjunto de actividades propuesto para cada iteración sea el mismo; en otro caso, es preciso especificar de manera concreta cada una de las iteraciones distintas.

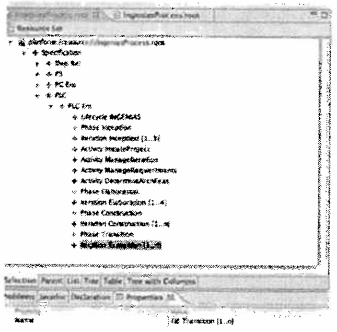

Fig. 2. Definición del Ciclo de vida INGENIAS con el editor

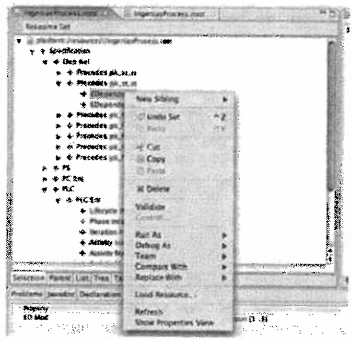

Fig. 3. Definición de Relaciones de Precedencia entre Fases con el editor

Utilizando el editor descrito en la sección anterior, la representación de este ciclo de vida iterativo e incremental se realiza en varios pasos. En primer lugar hay que definir los elementos que constituyen el ciclo de vida (fases, iteraciones, actividades) para lo cual se creará dentro de `Specification` un hijo de tipo `PLCProcess Lifecycle` y dentro de él, se creará una entidad de tipo `Lifecycle`, cuatro correspondientes a cada una de las fases de tipo `Phase` y una entidad de tipo `Iteration` que incluirá en el nombre el número mínimo y máximo admisible de iteraciones por cada fase especificada. A continuación se definen las relaciones temporales y de pertenencia existentes entre dichos elementos. En la Fig. 2 se presenta una imagen del editor realizando la inclusión de iteraciones para una fase del ciclo de vida, mientras que en la figura Fig. 3 se muestra como se incluyen las relaciones de precedencia entre ciclo de vida, fases e iteraciones mediante la creación de hijos `Dep Rel` de tipo `Precedes`.

El código resultante para los elementos que determinan el recorrido y las actividades descritas puede verse a continuación entre las marcas `<plc>` y `</plc>`. En cuanto a las relaciones de precedencia, el código se encuentra entre las marcas `<depEnt>` y `</depEnt>` cualificados con los valores `pk_st_st`, `pk_fn_st` y `pk_fn_fn` para indicar que la apertura de un elemento supone la apertura del relacionado, o el cierre de uno la apertura del correspondiente o que el cierre del primero supone el cierre del segundo respectivamente:

```
<?xml version="1.0" encoding="UTF-8"?>
<root:Specification xmi:version="2.0"
mlns:xmi="http://www.omg.org/XMI"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:dep_entities="dep_entities"  xmlns:pc_entities="pc_entities"
```



```
  xmlns:plc_entities="plc_entities" xmlns:ps_entities="ps_entities"
  xmlns:root="root">
...
  <depRel>
    <dep xsi:type="dep_relations:Precedes">
      <supplier EDMod="//@plc/@plcent/@wdef.1"/>
      <client EDMod="//@plc/@plcent/@wdef.0"/>
    </dep>
    <dep xsi:type="dep_relations:Precedes">
      <supplier EDMod="//@plc/@plcent/@wdef.2"/>
      <client EDMod="//@plc/@plcent/@wdef.1"/>
    </dep>
...
  <plc>
    <plcent>
      <wdef xsi:type="plc_entities:Lifecycle" name="INGENIAS"/>
      <wdef xsi:type="plc_entities:Phase" name="Inception"/>
      <wdef xsi:type="plc_entities:Iteration" name="Inception
[1..3]"/>
      ...
    </plcent>
  </plc>
</root:Specification>
```

Una vez finalizados los pasos anteriores hay que especificar para cada iteración a que fase pertenecen y cuales son las actividades que se desarrollarán. Por ejemplo, para la Fase de "Inception" del proceso descrito las actividades que hay que desarrollar en cada iteración son las que se muestran en la Fig. 4.

El proceso que se sigue para representar la secuencia de realización de estas actividades primarias es el descrito con anterioridad, introduciendo las actividades como entidades que aparecen especificadas entre marcas `<plc>` y `</plc>` en el código resultante y estableciendo las precedencias existentes como hijos `Dep Rel` de tipo `Precedes`. Finalmente, se indican las relaciones de pertenencia existentes entre las actividades y la iteración en que se realizan, y entre iteraciones y la fase en que se desarrollan; para ello, se debe crear un nuevo hijo de tipo `PS` (Procesos) y, dentro de él, otro correspondiente al tipo `PS Rel` (Relaciones entre Procesos). A continuación, se añaden relaciones de tipo `RParen W` y se selecciona la actividad y la iteración a la que pertenecen. El código que se obtiene en la especificación es semejante al anterior pero con marcas del tipo:

```
<?xml version="1.0" encoding="UTF-8"?>
...
    <psrel>
      <rparenw>
        <rpwsource rpwp="//@plc/@plcent/@wdef.1"/>
        <rpwtarget rpwp="//@plc/@plcent/@wdef.2"/>
      </rparenw>
...
</root:Specification>
```

Una vez especificado el recorrido del Proceso de desarrollo a implantar junto con las actividades que tienen que desarrollarse en cada una de las iteraciones es preciso describir las disciplinas implicadas en la formalización de estas actividades.

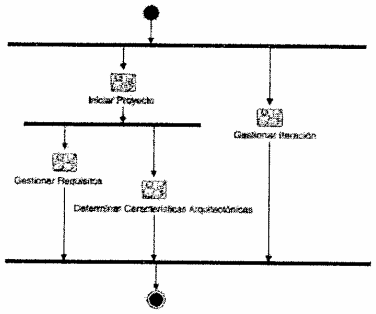

Fig. 4. Fase de "Inception".

4. Las disciplinas de INGENIAS

Las disciplinas dentro de SPEM determinan paquetes de procesos que permiten dividir las actividades de un proceso de acuerdo a un "Tema" común, es decir representan una especialización de las subactividades seleccionadas, de tal manera que estas nuevas subactividades no pueden aparecer en otras disciplinas o paquetes. En el caso de estudio utilizado las disciplinas que conforman el Proceso de Desarrollo son *RequirementSpecification, Analyse, Design* y *Implementation*. La definición de estas disciplinas se realiza creando dentro de `Specification` un hijo de tipo `PC Ent` (entidades de los *Process Components*) y dentro de él se incluyen las cuatro disciplinas mediante la selección de nuevas entidades de tipo `Discipline`.

Como continuación se sugiere que antes de detallar las subactividades (*tareas*) que conforman cada disciplina se proceda a la inclusión de los participantes en base a los roles que van a tener a lo largo del proyecto y de los productos que van a ser utilizados. En el caso de estudio seleccionado los roles son: *Analyst, Architect, Developer, Tester, Project Manager* y *FreeMan* y los productos que se van a manejar: *Use Case Diagram, Environment Model, Organization Model, Interaction Model, Agent Model, Object and Task Model, Glossary, NonFunctional Requirement* y *Stakeholder Vision*.

El mecanismo para incluir los roles y los productos con el editor es crear en `PS Ent` nuevas entidades de tipo `Proc Rol` y `Work Prod` respectivamente.

Una vez que están modelados los participantes, los productos, las disciplinas y el recorrido del proceso que se pretende especificar se procederá a descomponer cada actividad dentro de una iteración en base a tareas pertenecientes a una de las disciplinas seleccionadas y que deberá ser desarrollada por alguno de los participantes especificados consumiendo y/o produciendo alguno de los productos existentes. El proceso para realizar este paso es similar a los ya detallados pero con la diferencia de

que hay que modelar las tareas como `Step` dentro de `PS Ent` y de que las relaciones existentes entre las actividades y las tareas se modelan como una `PS Rel` de tipo `RAc Step`, y las de realización de actividades por parte de un participante mediante relaciones `PS Rel` de tipo `RAss`. En cuanto a las relaciones que se establecen entre las actividades y los productos consumidos y/o producidos, la versión v 1.1 presenta la limitación de no permitir definir una relación directa entre ellos, por lo que es necesario establecerla vía la disciplina en que se usa y el rol que lo maneja.

5. Conclusiones y Trabajo Futuro.

En este trabajo se muestra de forma práctica cómo utilizando un estándar de modelado se obtiene la definición del proceso de desarrollo asociado a una metodología orientada a agentes concreta. A partir de dicha especificación se esboza el modo de construir una herramienta de soporte al modelado de procesos usando el estándar de OMG, SPEM y herramientas, como Eclipse EMF, que facilitan la construcción del software necesario. El proceso anterior proporciona múltiples posibilidades, ya que simplifica enormemente la construcción de herramientas software de soporte a la definición de procesos. Además, el hecho de modelar el proceso de desarrollo de una metodología es fundamental para facilitar el aprendizaje de la misma y su utilización diaria.

Los resultados obtenidos permiten guiar al desarrollador de un MAS en los pasos a dar en su construcción, desde la definición de requerimientos hasta la implementación. De modo que una herramienta de soporte metodológico, que contemple el proceso definido, podrá guiar al desarrollador paso a paso, indicándole lo que debe hacer a continuación o comprobando, al realizar una definición, que se han cumplimentado los pasos previos que son necesarios. El trabajo desarrollado hasta el momento fundamenta la idea de que los aspectos antes reseñados son alcanzables en su totalidad. Sin embargo, es necesario continuar en esta línea para abordar otros apartados del modelado, para especificar múltiples metodologías según el estándar, para comprobar con un caso de uso real que el proceso modelado está completo, etc.

Otra importante línea de actuación en el futuro será usar los modelos de proceso de las diferentes metodologías, obtenidos ajustándose al estándar, como base para la comparación de las mismas. Este problema podría abordarse definiendo sobre el modelo medidas objetivas que permitan indicar si una metodología es mejor que otra, o más adecuada para un dominio concreto o incluso un MAS particular.

Referencias

[1] D. Amyot, H. Farah, and JF Roy. Evaluation of Development Tools for Domain-Specific Modeling Languages. Proceedings of the 5th Workshop on System Analysis and Modelling, 2006.

[2] Carole Bernon, Massimo Cossentino, and Juan Pavón. Agent-oriented software engineering. Knowl. Eng. Rev., 20(2):99–116, 2005.

[3] E.Breton and Jean Bezivin. Model driven process engineering. In Int. Computer Software and Applications Conf. (COMPSAC'2001), page 225, Chicago, Illinois, 2001.
[4] Frank Budinsky. Eclipse Modelling Framework: Developer's Guide. AddisonWesley, 2003.
[5] G. Caire, F. Garijo, J. Gómez, J. Pavon, F. Leal, P. Chainho, P. Kearney, J. Stark, R. Evans, and P. Massonet. Agent oriented analysis using MESSAGE/UML. In Agent-Oriented Software Engineering. Second International Workshop, AOSE 2001, volume 2222 of Lecture Notes in Computer Science. Springer-Verlag.
[6] Gerardo Canfora, Félix García, Mario Piattini, Francisco Ruiz, and C. A. Visaggio. Applying a framework for the improvement of software process maturity. Softw. Pract. Exper., 36(3):283–304, 2006.
[7] P. Cuesta, A. Gómez, J.C. González, and F. J. Rodríguez. The MESMA methodology for agent-oriented software engineering. In Proceedings of First International Workshop on Practical Applications of Agents and Multiagent Systems (IWPAAMS'2002), pages 87–98, 2002.
[8] J. J. Gómez-Sanz. Modelado de Sistemas Multi-Agente. PhD thesis, Departamento de Sistemas Informáticos y Programación, Universidad Complutense Madrid, 2002.
[9] J. J. Gómez-Sanz and R. Fuentes. Agent oriented software engineering with Ingenias. In Fourth Iberoamerican Workshop on Multi-Agent Systems (Iberagents'2002), a workshop of IBERAMIA'2002, the VIII Iberoamerican Conference on Artificial Intelligence, 2002.
[10] Ivar Jacobson, Grady Booch, and Jim Rumbaugh. The Unified Software Development Process. Addison-Wesley, 1999.
[11] Steven Kelly. GOPRR Description. PhD dissertation, Appendix 1, 1997.
[12] Ana Mas. Agentes Software y Sistemas Multi-Agentes. Pearson Prentice Hall, 2004.
[13] S. A. O'Malley and S. A. DeLoach. Determining when to use an agent-oriented software engineering paradigm. In M. Wooldridge, G. Wei[U+FFFD] and P. Ciancarini, editors, Agent-Oriented Software Engineering. Second International Workshop, AOSE 2001, volume 2222 of Lecture Notes in Computer Science. Springer-Verlag, 2002.
[14] Bill Moore et al. Eclipse Development using Graphical Editing Framework and the Eclipse Modelling Framework. IBM Redbooks, 2004.
[15] OMG. Meta Object Facility (MOF) Specification.
[16] OMG. Software Process Engineering Metamodel Specification. 2005.
[17] J. Pavón and J. Gómez-Sanz. Agent Oriented Software Engineering with INGENIAS. Multi-Agent Systems and Applications III, 2691:394–403, 2003.
[18] Juha-Pekka Tolvanen. GOPRR metamodeling language. 2000.

Diseño de Agentes Supervisores para el Control de Robots Móviles

Ignacio Benítez, José Luis Díez, Raúl Simarro y Pedro Albertos

Dpto. de Ingeniería de Sistemas y Automática
Universidad Politécnica de Valencia
{igbesan,jldiez}@isa.upv.es, rausifer@upvnet.upv.es, pedro@aii.upv.es

Abstract. El presente artículo recoge los primeros resultados en el diseño de agentes supervisores que cooperan en el control de robots móviles, mediante la partición del espacio de actuación en áreas de influencia distintas para cada agente, incorporando transiciones de tipo borroso en las zonas en que el robot móvil sigue una trayectoria prefijada pasando de un área de influencia a otra.

Keywords: Robótica móvil, Sistemas multiagente, Lógica borrosa.

1 Introducción

El presente trabajo se enmarca dentro de las actividades de investigación y desarrollo del proyecto KERTROL[1], el cual lleva por título "Núcleo de control en los sistemas empotrados fuertemente interconectados", y tiene por objetivo el estudio de la compatibilización del control distribuido con sistemas empotrados de tiempo real, teniendo en cuenta las nuevas tendencias que apuntan a una descentralización del control mediante la implementación de modelos basados en agentes [1] o en holones [2]. Otras líneas de investigación contemplan la definición de un *kernel* o núcleo de control que cumpla con las restricciones de los sistemas en tiempo real [3], o el estudio de la asignación de tiempos a tareas de control en sistemas empotrados [4]. La plataforma de desarrollo consiste en un entorno cerrado en el que robots móviles realizan tareas específicas (como el transporte de objetos), desde un punto objetivo a otro, en un diseño en que uno o más procesadores externos coordinan sus esfuerzos para controlar las trayectorias de los robots, evitando situaciones de colisión. El objetivo es el de particionar el espacio físico por el que se mueven los robots en áreas de influencia de distintos agentes supervisores, de forma que cuando un robot entra dentro del área de influencia de un agente supervisor, éste asume el control de ese robot, liberándose los demás agentes de esta tarea. Para ello se definen funciones de pertenencia borrosas [5] para todo el espacio de ejecución, vinculando cada función de pertenencia definida a un agente supervisor. Partiendo de trabajos en los que se investiga la dinámica de modelos basados en agentes móviles [6],[7], en este caso los agentes se mantienen fijos y ejercen tareas de supervisión y control sobre las unidades móviles que se encuentren dentro de sus áreas de pertenencia.

[1] MEC: DPI2005-09327-C02-01/02