

Patrón de requisito basado en dependencias de plan en i^* para detectar proactividad en sistemas informáticos

Alain Pérez Acosta

correo electrónico: aperezac@ceis.cujae.edu.cu

Instituto Superior Politécnico José Antonio Echeverría, Cujae, La Habana, Cuba

Artículo Original

Mailyn Moreno Espino

correo electrónico: my@ceis.cujae.edu.cu

Instituto Superior Politécnico José Antonio Echeverría, Cujae, La Habana, Cuba

Resumen

El objetivo del trabajo es presentar un patrón de requisito, basado en modelos de i^* , que permite detectar proactividad en sistemas informáticos desde la etapa de Requisitos. El patrón obtenido como resultado de esta investigación, permite detectar proactividad cuando la dependencia que se establece entre los actores involucrados es de plan, y además uno de los actores tiene intenciones que denotan un futuro comportamiento proactivo en el software. Para validar el patrón se realizó un estudio de casos, tomando como lógica de análisis el desarrollo de un *dashboard* proactivo para apoyar la toma de decisiones en una facultad universitaria. Basado en los resultados del estudio de casos, se puede concluir que el patrón propuesto permitió modelar las dependencias intencionales entre los actores, detectar un comportamiento proactivo y delegar la proactividad en el sistema de software a desarrollar.

Palabras claves: proactividad, patrón de requisito, i^* , sistemas informáticos

Recibido: 17 de enero del 2015 Aprobado: 25 de noviembre del 2015

INTRODUCCIÓN

Según plantea Víctor Frankl en [1], la proactividad es “una actitud en la que el sujeto asume el pleno control de su conducta vital de modo activo, lo que implica la toma de iniciativas en el desarrollo de acciones creativas y audaces para generar mejoras”. La proactividad se trata de tomar el control para hacer que las cosas sucedan, no esperar a que ocurran eventualmente [2].

La proactividad en un contexto informático es cuando un software es capaz de exhibir un comportamiento dirigido a metas, tomando la iniciativa con el fin de satisfacer sus metas de diseño [3].

Hasta ahora en la Informática el término proactividad ha sido tratado mayormente por los trabajos relacionados con el paradigma de agentes [4-6]. La proactividad es una de las características más distintivas de los agentes [6, 7], es un comportamiento donde el agente trabaja para alcanzar una meta. El comportamiento proactivo permite que se le deleguen las metas al software y este trabaje para cumplirlas. Un software con comportamiento proactivo en áreas como la gestión, la toma de decisiones [8, 9], los ambientes asistidos [10, 11] y los sistemas en tiempo real [12], es un beneficio para el usuario final que lo utiliza.

A pesar de los avances del paradigma de agentes, las metodologías para el desarrollo de software orientado a agentes se enfocan en tratar problemas como la comunicación entre los agentes y la organización [13]. No se enfocan en cómo modelar la proactividad, cómo descubrirla desde los requisitos tempranos de un software.

La proactividad, tanto en los humanos como en los agentes, parte de una representación de las metas a cumplir [3]. En el ámbito de esta representación de metas se hace importante el uso de patrones. Christopher Alexander [14] expone que "Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, entonces describe el núcleo de la solución para ese problema, de manera tal que usted pueda utilizar esta solución un millón de veces, sin tener que hacerlo dos veces de la misma forma". Uno de los contextos donde más se ha trabajado el desarrollo de patrones es en el orientado a objetos. Los patrones de diseño orientado a objetos son los más conocidos [15].

En la orientación a agentes también se han concebido patrones que permitan avanzar en el desarrollo de sistemas orientados a agentes. Se ha trabajado en patrones para desarrollar agentes móviles [16], para la comunicación entre agentes [17, 18] y para la organización de un sistema de agentes [19]. Estos patrones estudiados en la orientación a agentes no hacen énfasis en la proactividad o ambientes a observar, sino en otras propiedades como la cooperación, la comunicación y la estructura organizacional de los agentes.

En este sentido, existen trabajos como [20] que proponen patrones para detectar proactividad en sistemas informáticos durante la etapa de requisitos. Los patrones propuestos en [20] se basan en modelos de i* [21, 22] que cumplen con las siguientes condiciones: actores en la etapa de *Requisitos Tempranos* que denotan un comportamiento proactivo en los *Requisitos Tardíos*. Yu en [23] define un patrón de i* como "un modelo que constituye una generalización de un dominio específico o una situación de interés, que puede ser contextualizado cuando se aplica a una situación más específica".

Los patrones *Hard Goal Why Dependency* y *Resource Why Dependency* que se proponen en [20] se pueden usar cuando la dependencia entre dos actores es de meta fuerte y recurso, respectivamente. Sin embargo, en i* también se pueden establecer dependencias de plan entre los actores [21, 22]. Para ese caso, los patrones propuestos en [20] no se pueden utilizar.

Por tanto, el objetivo del presente trabajo es proponer un patrón de requisito, basado en modelos de i*, para cuando la dependencia entre dos actores es de plan y uno de los actores tiene intenciones que denotan un futuro comportamiento proactivo en el software. El patrón que se obtiene como resultado del trabajo se denomina *Task Why Dependency* y trata con el siguiente problema: actores en los *Requisitos Tempranos* con intenciones que denotan un comportamiento proactivo. Para validar el patrón *Task Why Dependency* se realizó un estudio de caso [24] donde el objeto de análisis es el desarrollo de un *dashboard* proactivo [25-28]. A pesar de que en las

condiciones generales del patrón se plantea que se debe utilizar en la etapa de *Requisitos Tempranos*, el estudio de casos realizado reveló que se puede usar para refinar los modelos obtenidos en los *Requisitos Tardíos*.

MATERIALES Y MÉTODOS

Modelado social y lenguaje i*

La Ingeniería de Requisitos es un área de la Ingeniería de Software que va más allá de definir la funcionalidad esperada del sistema de software a desarrollar, puesto que establece la relación entre esta funcionalidad y los procesos de negocio de la empresa. La Ingeniería de Requisitos facilita el mecanismo apropiado para comprender lo que quiere el cliente (*stakeholder*), analizando necesidades, confirmando su viabilidad y negociando una solución razonable [29].

La captura de requisitos es la actividad mediante la cual el equipo de desarrollo de un sistema de software extrae, de cualquier fuente de información disponible, las necesidades que debe cubrir dicho sistema [30]. La captura de requisitos incluye varias actividades en las cuales la interacción con los clientes tiene una importancia primordial [31].

En la Ingeniería de Software y de Sistemas de Información la construcción de modelos mayormente ha girado en torno a las relaciones estáticas y las propiedades dinámicas y de comportamiento de los mismos [22]. Este enfoque es obvio, ya que los modelos conceptuales al final se traducen en los datos y las operaciones que ejecutará la computadora. Sin embargo, un sistema para tener éxito debe funcionar dentro del contexto de su entorno [21].

Adoptando una visión social del mundo, se puede ver que en este existe la intencionalidad. La intencionalidad la originan los actores, como los seres humanos. Los actores intencionales tienen necesidades y deseos, y realizan acciones para tratar de satisfacerlos. Los actores pueden elegir qué acciones tomar, lo cual los hace autónomos. Los actores no existen de forma aislada, existen en algún entorno compartiendo e interactuando con otros [21].

El modelado social, al enfocarse en la etapa temprana de la Ingeniería de Requisitos, se centra en la dimensión social de los sistemas informáticos y su entorno. En un enfoque social, los intereses estratégicos de los actores deben ser utilizados para guiar la búsqueda de concepciones alternativas para el nuevo sistema. Cada actor debe proponer sus intereses estratégicos [21]. El modelado social ve la Ingeniería de Requisitos de una forma orientada a metas. Un análisis de metas revela deseos, lo que permite identificar conflictos o expectativas. Un modelo orientado a metas puede ayudar a gestionar cambios. Las metas proporcionan criterios y guías para generar y evaluar posibles soluciones [32].

El lenguaje de modelado i* introduce aspectos del modelado social y del razonamiento sobre los métodos de Ingeniería de Sistemas de Información, especialmente a nivel de Requisitos [21]. i* reconoce la primacía de los actores sociales, los actores son vistos como intencionales,

es decir, tienen metas, creencias, habilidades y compromisos. El análisis se enfoca en la captura de las metas de los distintos actores, dada la configuración de las relaciones entre los actores humanos y el sistema. La configuración de estas relaciones puede ayudar a plasmar los intereses estratégicos de los actores [22].

En i^* se propone dividir la captura de requisitos en dos fases: *Análisis de Requisitos Tempranos* y *Análisis de Requisitos Tardíos*. La primera fase consiste en la identificación y análisis de los principales actores del dominio involucrado en el problema, y de sus necesidades e intenciones. En el *Análisis de Requisitos Tardíos* se trata de modelar lo más claro posible “qué” debe hacer el futuro sistema [22]. Además i^* hace uso de dos modelos, cada uno con un nivel diferente de abstracción: el nivel intencional, representado por el Modelo de Dependencia Estratégica (*Strategic Dependence Model*, SD) y el nivel racional, representado por el modelo Estratégico de Racionalidad (*Strategic Rational Model*, SR) [22]. En el modelo SD se representan los actores y las distintas relaciones entre ellos, mientras que en el modelo SR de cada actor se representan las dependencias entre objetos dentro del actor.

Patrones basados en modelos de i^* para detectar proactividad

Para detectar la proactividad se necesita conocer las causas de las dependencias entre los actores (los porqués), es decir, la intencionalidad de las relaciones. A partir de los modelos de i^* que se obtienen en la etapa de *Requisitos Tempranos*, es posible detectar metas que se pueden cumplimentar de forma proactiva [20]. Según Yu [33] la dependencia es intencional si el objeto del que se depende (*dependum*) está relacionado de alguna manera a la meta o deseo del actor que depende (*dependor*). Las intenciones se pueden delegar al software y de esta forma que el mismo trabaje en pos de lograr metas e intenciones, lo que conlleva a un comportamiento proactivo [20].

En [20] se presentan dos patrones de requisitos que utilizan como base los modelos de i^* en las etapas de *Requisitos Tempranos* y *Requisitos Tardíos*. Estos dos patrones tratan con un problema común: actores en los *Requisitos Tempranos* con intenciones que denotan un comportamiento proactivo. También se describe la solución para delegar estas intenciones en el software que se desarrollará en los *Requisitos Tardíos*. Los patrones propuestos en [20] se denominan *Hard Goal Why Dependency* y *Resource Why Dependency*. El patrón *Hard Goal Why Dependency* es aplicable cuando existe una dependencia de meta fuerte entre los actores involucrados. De forma análoga, el patrón *Resource Why Dependency* es aplicable cuando existe una dependencia de recurso entre los actores. Una descripción más detallada de ambos patrones se puede encontrar en [20]. Sin embargo, ninguno de los dos patrones abarca el caso cuando la dependencia que se establece entre los actores es de plan.

Estudio de casos

Para validar el patrón que se propone como resultado de este trabajo se utilizó un estudio de casos. El estudio de casos constituye un método importante y reconocido para la investigación [24, 34]. En el caso particular de la Informática se ha usado y defendido su uso en [34, 35] en situaciones que son difíciles de reproducir en un experimento de laboratorio en que es fácilmente aislable y controlable el contexto. El estudio de casos permite estudiar más a fondo una menor cantidad de experiencias diferentes (entidades o unidades) que se presentan en su complejidad. La presentación de patrones de diseño o de implementación, muchas veces ha seguido la lógica de presentar los casos que muestran la aplicabilidad de la propuesta, para tratar de aprender y generalizar de ellos [24, 35].

De las seis estructuras de presentación de los casos [24] se ha optado por la estructura lineal-analítica, que es útil para todos los casos y es la más usada. En ella se presenta el aspecto o problema estudiado y revisa la literatura relevante, sigue con los métodos, los hallazgos a partir de los datos recolectados, y luego las conclusiones e implicaciones. Siguiendo las recomendaciones de [24, 34], los casos se presentan haciendo explícitos algunos elementos importantes de este método.

- Pregunta(s) de estudio y proposiciones: Se aclara la pregunta en que se enfocará cada caso, como orientación de la investigación en el caso. Se incluye las proposiciones a verificar en cada caso.
- Contexto: Se presenta el contexto del caso, comparando con otros trabajos de la literatura y se justifica la relevancia del caso. Se expone la unidad de análisis de cada caso.
- Lógica de análisis: Se explica la lógica que relaciona los datos con las proposiciones. En esta sección, la fuente de la información presentada sería la observación directa o la observación participativa según la clasificación de [24].
- Discusión: Se explican los criterios para interpretar los hallazgos. En cada caso el énfasis radica en observar cómo las propuestas conducen a los resultados esperados de una manera que puede ser repetida luego en otras situaciones, y cómo esto supera a las alternativas existentes; así como justificar las condiciones que permiten generalizar el uso de las propuestas teniendo en cuenta cómo las condiciones de cada experimento limitan o no esta generalización.

RESULTADOS

Como resultado de este trabajo se presenta el patrón *Task Why Dependency*, que tiene como base los modelos de i^* en los *Requisitos Tempranos* y *Tardíos*. Este patrón trata con el siguiente problema: actores en los *Requisitos Tempranos* con intenciones que denotan un comportamiento proactivo. La solución que se describe para delegar estas intenciones en el software que se desarrollará se ve en los *Requisitos Tardíos*. El patrón

Task Why Dependency se puede utilizar cuando se desea desarrollar un software con un comportamiento proactivo y existe una dependencia de plan entre actores estratégicos del proceso que se quiere automatizar. A continuación se explican los elementos fundamentales del patrón:

Nombre del patrón: *Task Why Dependency*.

Problema: Se está analizando un modelo de *Requisitos Tempranos* plasmados en el lenguaje i* con el objetivo de detectar dependencias intencionales y se presenta el estado que se describe a continuación:

Existe un actor "Beneficiario" que tiene una meta fuerte "Meta principal" (siguiendo el ejemplo de la figura 1). A dicha meta le contribuye de forma positiva la meta suave "Intención". Ambas metas son propias del actor, es decir, no se derivan de una dependencia. Existe una dependencia entre el actor "Beneficiario" y el actor "Sistema" de un plan "Plan 1" para cumplimentar la meta fuerte "Meta principal" del actor "Beneficiario". El plan, "Plan 1," es un medio para cumplir la meta fuerte "Meta fuerte 2" del actor "Sistema".

El actor "Sistema" para poder cumplimentar el plan "Plan 1" necesita del recurso "Recurso 1" del actor "Beneficiario".

El recurso, "Recurso 1", constituye un medio para cumplir la meta suave "Intención".

Solución: Para delegar las intenciones en el software que se pretende desarrollar se construye un modelo o subconjunto de modelos de *Requisitos Tardíos* en i* donde se debe:

Transformar el actor "Sistema" en un actor con estereotipo de agente.

Delegar al actor "Sistema" la meta suave "Intención", que representa la intención que denota un comportamiento proactivo.

Reflejar que la meta suave "Intención" contribuye de forma positiva a lograr la meta fuerte "Meta fuerte 2" del actor "Sistema".

Representar un plan "Plan 2" como un medio para cumplimentar la meta suave "Intención".

Delegar el recurso "Recurso 1" en el actor "Sistema".

Reflejar que el recurso "Recurso 1" es un medio para cumplir "Plan 1" y "Plan 2".

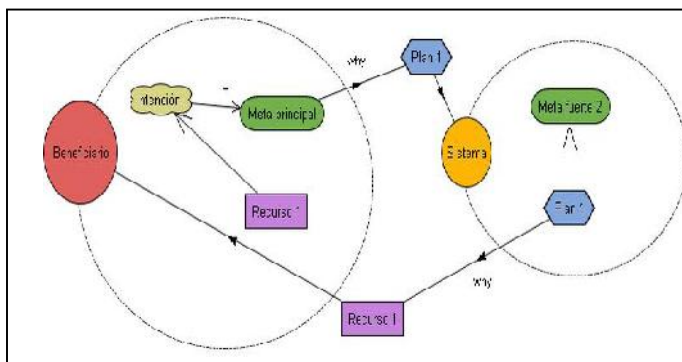


Fig. 1. Subconjunto del modelo de Requisitos Tempranos con i* que representa el problema del patrón Task Why Dependency.

La figura 2 muestra la solución del patrón Task Why Dependency plasmada en un subconjunto de los modelos de *Requisitos Tardíos* de i*. Es importante destacar que las relaciones y entidades presentes en los *Requisitos Tempranos* que no forman parte del subconjunto abordado en el problema del patrón Task Why Dependency se reflejarán en los *Requisitos Tardíos* siguiendo lo que establece i*.

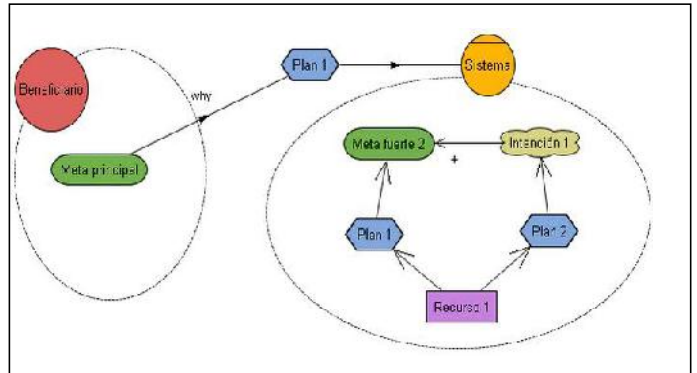


Fig. 2. Subconjunto del modelo de Requisitos Tardíos con i* que representa la solución del patrón Task Why Dependency.

Consecuencias: Este patrón permite delegar en el software intenciones para que este cumplimente sus metas de forma proactiva. Su objetivo es proporcionar una vía para delegar dichas intenciones cuando se desea tener proactividad en un software que pueda reportar beneficios en su uso. El plan que representa el medio para cumplimentar la intención hay que implementarlo para que pueda haber proactividad.

DISCUSIÓN

Para validar el patrón *Task Why Dependency* se presenta un estudio de caso. La unidad de análisis es el desarrollo de un *dashboard* proactivo para el apoyo a la toma de decisiones. El caso fue escogido por su relevancia, siguiendo las recomendaciones de [24]. El propósito es la validación del patrón *Task Why Dependency*, por lo que puede considerarse como explicativo.

Preguntas de estudio y proposiciones

¿Cómo el patrón de requisitos *Task Why Dependency* ayuda a identificar requisitos proactivos?

Las proposiciones correspondientes son:

- La utilización del lenguaje i* y el patrón *Task Why Dependency* de requisitos para detectar proactividad permiten identificar requisitos proactivos.

Contexto

La unidad de análisis es el desarrollo de un *dashboard* proactivo para el apoyo a la toma de decisiones. Según [26] un *dashboard* es una herramienta visual e interactiva de función administrativa que muestra en una pantalla la información más relevante para lograr una o varias metas individuales o empresariales, permitiendo a los usuarios identificar, explorar y comunicar áreas de problema que necesitan acción correctiva. Dentro de las metas más importantes que deben tener los *dashboards* se encuentran [27]:

- Responder a las preguntas fundamentales acerca del dominio del negocio.

- Alertar al usuario cuando ocurran problemas que afecten el negocio, como por ejemplo una disminución drástica en las ventas de un producto.
- Ayudar en la toma de decisiones a los usuarios.

Si unido a las metas anteriores se toma además en cuenta que según [8, 9], un software con comportamiento proactivo en áreas como la toma de decisiones representa un beneficio para el usuario final que lo utiliza, sería útil que los *dashboards* tuvieran un comportamiento proactivo orientado a las metas de sus usuarios. Por tanto, un caso a tener en cuenta la proactividad es en la construcción de un *dashboard*.

Lógica de análisis

En la lógica de análisis se abordarán la pregunta de estudio, la proposición y cómo estas serán respondidas.

¿Cómo el patrón de requisitos *Task Why Dependency* ayuda a identificar requisitos proactivos?

En una facultad universitaria se desea monitorear el proceso docente para apoyar la toma de decisiones. Partiendo de las ventajas que presentan las herramientas de soporte a la toma de decisiones, se determinó construir un *dashboard* para monitorear el proceso docente. La figura 3 expone los *Requisitos Tempranos*, modelados con *i**, de las metas y relaciones de los actores estratégicos que se involucran en el proceso, siguiendo los modelos que se proponen en [36].

El actor “Vicedecano Docente” tiene una meta estratégica: “Aumentar promoción de estudiantes”. Para cumplimentar dicha meta el Vicedecano Docente necesita que el actor “Diseñador de *dashboard*” cumpla la meta “Diseñar *dashboard*”. En esta relación de dependencia se puede comprender el porqué existe la necesidad de la meta “Diseñar *dashboard*” expresado con la palabra *why*. La meta estratégica “Aumentar promoción de estudiantes” se descompone en dos metas de decisión: “Disminuir cantidad de estudiantes suspensos” y “Disminuir cantidad de estudiantes invalidados”. A su vez, la meta “Analizar resultados docentes” constituye el medio para lograr cumplir la meta “Disminuir cantidad de estudiantes suspensos”.

De forma similar ocurre con las metas “Analizar resultados de asistencias” y “Disminuir cantidad de estudiantes invalidados”. Para cumplir las metas “Analizar resultados docentes” y “Analizar resultados de asistencia” se deben realizar los planes “Analizar resultados docentes por curso” y “Analizar asistencia de los estudiantes a clases” respectivamente. Además el Vicedecano Docente tiene la intención de “Mantenerse informado de la promoción”, que contribuye de forma positiva a lograr la meta estratégica.

A su vez el Diseñador de *dashboard* para cumplir la meta “Diseñar *dashboard*” necesita que el actor “Analista de *dashboard*” cumpla las metas “Obtener KPIs” y “Obtener requisitos”. El Analista de *dashboard* depende del Vicedecano Docente para obtener los “Requisitos no funcionales”. Este recurso es el medio para cumplir el plan “Obtener requisitos no funcionales”.

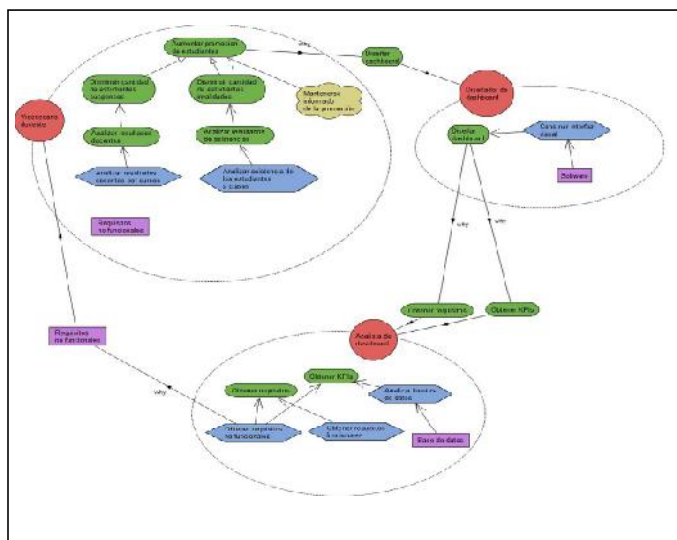


Fig. 3. Modelo de Requisitos Tempranos del dashboard.

En la figura 4 se presentan los modelos obtenidos en la etapa de *Requisitos Tardíos* siguiendo las recomendaciones de [36].

En la etapa de *Requisitos Tardíos* desaparecen los actores “Analista de *dashboard*” y “Diseñador de *dashboard*” y aparece el actor “*Dashboard*” con el estereotipo de sistema que propone *i**. La meta principal de este actor es “Monitorear promoción”. Para cumplir esa meta necesita ejecutar el plan “Visualizar KPIs”. El plan “Visualizar KPIs” constituye el medio para lograr la meta “Aumentar promoción de estudiantes” del Vicedecano Docente. Para cumplir el plan “Visualizar KPIs” el actor *Dashboard* necesita conocer las métricas que al Vicedecano Docente le interesa, por lo que depende del recurso “KPIs” del Vicedecano Docente. El recurso “KPIs” a su vez constituye un medio para lograr la intención “Mantenerse informado de la promoción”.

Aunque los modelos de la figura 4 se corresponden con la etapa de *Requisitos Tardíos*, no denotan un comportamiento proactivo en el actor *Dashboard*, debido a que la intención “Mantenerse informado de la promoción” permanece en el actor Vicedecano Docente. Por tanto, al hacer un análisis de los modelos de la figura 4 siguiendo lo que plantea el patrón *Task Why Dependency*, se puede comprobar que:

Existe un actor Vicedecano Docente (que representa un humano) que tiene una meta fuerte “Aumentar promoción de estudiantes”. A esta meta le contribuye de forma positiva la meta suave “Mantenerse informado de la promoción”, dicha meta suave representa una intención. Ambas metas son del actor Vicedecano Docente, es decir, no se derivan de una dependencia.

Para cumplir la meta “Aumentar promoción de estudiantes” el Vicedecano Docente tiene una dependencia con el actor *Dashboard* (que representa un sistema) del plan “Visualizar KPIs”.

El actor *Dashboard* para implementar el plan “Visualizar KPIs” depende del recurso “KPIs” del Vicedecano Docente.

El recurso “KPIs” constituye un medio para lograr la intención “Mantenerse informado de la promoción”.

En la figura 5 se muestra el modelo de *Requisitos Tardíos* una vez aplicado el patrón *Task Why Dependency*.

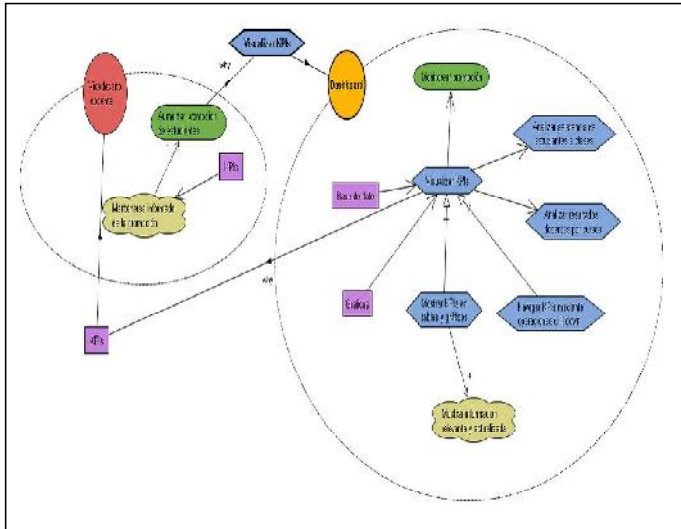


Fig. 4. Modelos de Requisitos Tardíos.

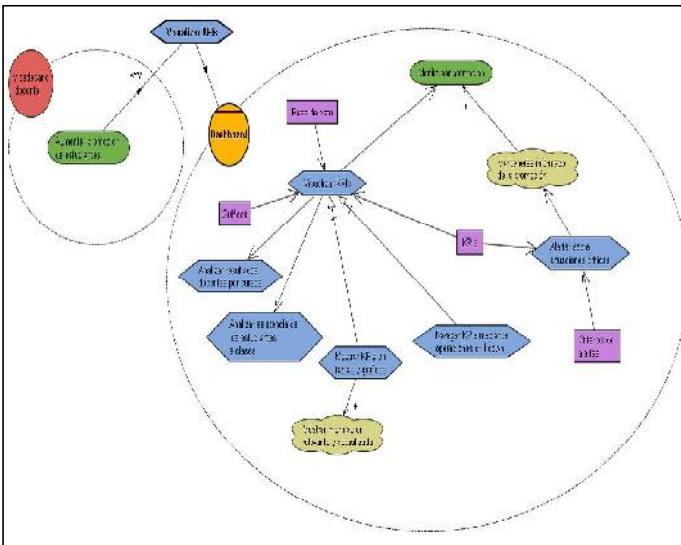


Fig. 5. Modelos de Requisitos Tardíos con el patrón *Task Why Dependency*.

Teniendo en cuenta la solución propuesta por el patrón *Task Why Dependency*, los modelos de *Requisitos Tardíos* reflejarán los siguientes cambios:

El actor *Dashboard* se representa con el estereotipo de agente que propone i*.

La meta suave “Mantenerse informado de la promoción” que denota un comportamiento proactivo se le delega al actor *Dashboard*.

Representar que la meta suave “Mantenerse informado de la promoción” contribuye de forma positiva a lograr la meta fuerte “Monitorear promoción” del actor *Dashboard*.

Representar el plan “Alertar sobre situaciones críticas” como un medio para cumplir la meta suave “Mantenerse informado de la promoción”.

Delegar el recurso “KPIs” en el actor *Dashboard*.

Representar el recurso “KPIs” como un medio para cumplir los planes “Alertar sobre situaciones críticas” y “Visualizar KPIs”.

CONCLUSIONES

De los resultados obtenidos en el trabajo y de su discusión, se pueden obtener las siguientes conclusiones: 1. El patrón *Task Why Dependency* permite detectar proactividad en los sistemas informáticos en la etapa de *Requisitos* cuando existen intenciones entre los actores que denotan un comportamiento proactivo y una dependencia de plan entre los actores; 2. El estudio de casos permitió validar el patrón *Task Why Dependency* y demostró que es posible aplicar el patrón para refinar los modelos de *Requisitos Tardíos* y no necesariamente en la etapa de *Requisitos Tempranos* como se plantea en las condiciones iniciales del patrón.

REFERENCIAS

1. FRANKL, VIKTOR. *Man's Search for Meaning*. 2006, Beacon Press, 165 pp. ISBN 9780807014271.
2. GRANT, A. M.; ASHFORD, S. J. "The dynamics of proactivity at work". *Research in Organizational Behavior*. 2008, vol. 28, pp. 3-34.
3. JENNINGS, NICHOLAS R. "An agent-based approach for building complex software systems". *Communications of the ACM*. 2001, vol. 44, núm.4, pp. 35-41. ISSN: 0001-0782.
4. COSSENTINO, MASSIMO; GLEIZES, MARIE-PIERRE; MOLESINI, AMBRA; OMICINI, ANDREA. Processes Engineering and AOSE. En *Agent-Oriented Software Engineering X*. Springer Berlin Heidelberg, 2011, pp. 191-212. ISBN: 978-3-642-19207-4.
5. VAN DER HOEK, WIEBE; WOOLDRIDGE, MICHAEL. Multi-Agent Systems. En *Handbook of Knowledge Representation*. Elsevier, 2008, pp. 887-928.
6. WOOLDRIDGE, MICHAEL. *An Introduction to MultiAgent Systems*. 2009, John Wiley & Sons, 484 pp. ISBN: 978-0470519462.
7. O'MALLEY, SCOTT A.; DELOACH, SCOTT A. Determining When to Use an Agent-Oriented Software Engineering Paradigm. En *Agent-Oriented Software Engineering II*. Springer Berlin Heidelberg, 2002, pp. 188-205.
8. LI, JIAO; FENG, YUQIANG. Construction of Multi-Agent System for Decision Support of Online Shopping. En *Emerging Research in Artificial Intelligence and*

- Computational Intelligence*. Springer Berlin Heidelberg, 2011, pp. 318-324. ISBN: 978-3-642-24281-6.
9. **SRINIVASAN, S.; SINGH, JAGJIT; KUMAR, VIVEK.** "Multi-agent based decision Support System using Data Mining and Case Based Reasoning". *International Journal of Computer Science Issues*. 2011, vol. 8, núm.4, pp. 340-349.
10. **LINDGREN, HELENA; SURIE, DIPAK; NILSSON, INGEBORG.** Agent-Supported Assessment for Adaptive and Personalized Ambient Assisted Living. En *Trends in Practical Applications of Agents and Multiagent Systems*. Springer Berlin Heidelberg, 2011, pp. 25-32. ISBN: 978-3-642-19930-1.
11. **MCNAULL, J.; AUGUSTO, J.C.; MULVENNA, M.; MCCULLAGH, P.** "Multi-agent System Feedback and Support for Ambient Assisted Living". En *Actas de 8th International Conference on Intelligent Environments*, 2012.
12. **GHORBANI, J.; CHOUDHRY, M. A.; FELIACHI, A.** "Real-time multi agent system modeling for fault detection in power distribution systems ". En *Actas de North American Power Symposium*, 2012.
13. **MORANDINI, MIRKO; NGUYEN, DUYCU et al.** "Tool-Supported Development with Tropos: The Conference Management System Case Study". En *Agent-Oriented Software Engineering VIII*. Springer Berlin Heidelberg, 2008, pp. 182-196. ISBN: 978-3-540-79487-5.
14. **ALEXANDER, CHRISTOPHER; ISHIKAWA, SARA; SILVERSTEIN, MURRAY.** *A Pattern Language: Towns, Buildings, Construction*, New York, 1977, Oxford University Press. ISBN: 978-0195019193.
15. **GAMMA, ERICH; HELM, RICHARD et al.** *Design Patterns: Elements of Reusable Object-oriented Software*. 2004, Pearson Education, 395 pp. ISBN 978-8131700075.
16. **ARIDOR, YARIV; LANGE, DANNY B.** "Agent design patterns: elements of agent application design". En *Actas de Second International Conference on Autonomous Agents*, 1998.
17. **OLUYOMI, AYODELE; KARUNASEKERA, SHANIKA; STERLING, LEÓN.** "An Agent Design Pattern Classification Scheme: Capturing the Notions of Agency in Agent Design Patterns". En *Actas de 1th Asia-Pacific Software Engineering Conference*, 2004.
18. **SAUVAGE, SYLVAIN.** Design Patterns for Multiagent Systems Design. En *Advances in Artificial Intelligence*. Springer Berlin Heidelberg, 2004, pp. 352-361. ISBN: 978-3-540-21459-5.
19. **SABATUCCI, LUCA; COSENTINO, MASSIMO; GAGLIO, SALVATORE.** "A Semantic Description For Agent Design Patterns". En *Actas de Sixth International Workshop From Agent Theory to Agent Implementation*, International Joint Conference on Autonomous Agents and Multi-Agent Systems, 2008.
20. **MORENO, MAILYN.** "Patrones para incorporar proactividad en sistemas informáticos". Director: A. Rosete, M. Delgado, J. Pavón. Tesis Doctoral, Instituto Superior Politécnico José Antonio Echeverría, La Habana, 2013.
21. **YU, ERIC.** Social Modeling and i*. En *Conceptual Modeling: Foundations and Applications*. Springer-Verlag, 2009, pp. 99-121. ISBN: 978-3-642-02462-7.
22. **YU, ERIC; GIORGINI, PAOLO et al.** *Social Modeling for Requirements Engineering*, Cambridge. 2011, The MIT Press. ISBN: 9780262240550.
23. **STROHMAIER, MARKUS; HORKOFF, JENNIFER, et al.** Can Patterns Improve i* Modeling? Two Exploratory Studies. En *Requirements Engineering: Foundation for Software Quality*. Berlin: Springer Berlin Heidelberg, 2008, pp. 153-167. ISBN: 978-3-540-69060-3.
24. **YIN, ROBERT K.** *Case Study Research: Design and Methods*. 2003, SAGE Publications.
25. **LEMPINEN, HEIKKI.** "Constructing a Design Framework for Performance Dashboards". En *Nordic Contributions in IS Research*. Springer Berlin Heidelberg, 2012, pp. 109-130. ISBN: 978-3-642-32269-3.
26. **YIGITBASIOGLU, OGAN M.; VELCU, OANA.** "A review of dashboards in performance management: Implications for design and research". *International Journal of Accounting Information Systems*. 2011, vol. 13, pp. 19.
27. **ZAGORECKI, ADAM; RISTVEJ, JOZEF et al.** "Executive Dashboard Systems for Emergency Management". *COMMUNICATIONS*. 2012, vol. 4, núm.2, pp. 82-89.
28. **ZHANG, XIAONI; GALLAGHER, KEVIN; GOH, SAMUEL.** "BI Application: Dashboards for Healthcare". En *Actas de Seventeenth Americas Conference on Information Systems*, 2011.
29. **SOMMERVILLE, IAN.** "Integrated Requirements Engineering: A Tutorial". *IEEE Software*. 2005, vol. 22, núm.1, pp. 16-23.
30. **JACOBSON, I.; BOOCH, G.; RUMBAUGH, J.** *The Unified Software Development Process*, 2012, Prentice Hall, 512 pp. ISBN: 978-0321822000.
31. **FUENTES-FERNÁNDEZ, RUBÉN; GÓMEZ-SANZ, JORGE J.; PAVÓN, JUAN.** "Understanding the human context in requirements elicitation". *Requirements Engineering*. 2010, vol. 15, núm.3, pp. 267-283. ISSN: 0947-3602.
32. **HORKOFF, JENNIFER; YU, ERIC.** "Comparison and evaluation of goal-oriented satisfaction analysis techniques". *Requirements Engineering*. 2013, vol. 18, núm.3, pp. 199-222. ISSN: 0947-3602.
33. **YU, ERIC.** "Modelling Strategic Relationships for Process Reengineering". Tesis Doctoral, University of Toronto, Toronto, Canada, 1995.
34. **DUBÉ, LINE; PARÉ, GUY.** "Rigor in information systems positivist case research: current practices, trends, and recommendations". *MIS Quarterly*. 2003, vol. 27, núm.4, pp. 597-635.

35. CHEN, HONG-MEI; KAZMAN, RICK; PERRY, OPAL.

"From Software Architecture Analysis to Service Engineering: An Empirical Study of Methodology Development for Enterprise SOA Implementation". *IEEE Transactions On Services Computing*. 2010, vol. 3, núm.2, pp. 145-160.

36.ACOSTA, ALAIN PÉREZ; ESPINO, MAILYN

MORENO. "Modelos de requisitos basados en i* para detectar proactividad en dashboards". *Lámpsakos*. 2014, núm.12, pp. 101-109. ISSN: 2145-4086.

AUTORES

Alain Pérez Acosta

Ingeniero Informático, Instructor, Facultad de Ingeniería Informática, Instituto Superior Politécnico José Antonio Echeverría, Cujae, La Habana, Cuba

Mailyn Moreno Espino

Ingeniera Informática, Doctora en Ciencias Técnicas, Profesora Auxiliar, Facultad de Ingeniería Informática, Instituto Superior Politécnico José Antonio Echeverría, Cujae, La Habana, Cuba

Requirement Pattern Based on Dependencies of Plan in i* for Detecting Proactivity in Information-Technology Systems

Abstract

This paper aims to present a requirement pattern based on i*'s models that allows detecting proactivity in information-technology systems from the Requirements' phase. The pattern obtained as a result of this paper allows detecting proactivity when there is a plan's dependence established between the actors involved and in addition one of the actors has intentions that denote a future proactive behavior in the software. In order to validate the pattern a case study was performed taking as logic of analysis the development of a proactive dashboard to support the decision making in a college faculty. Based on the results of the case study, it can be concluded that the proposed pattern allowed modeling the intentional dependencies between the actors, detecting a proactive behavior and delegating the proactivity in the system of software to be developed.

Key words: proactivity, requirement pattern, i*, information-technology systems