**Instituto Superior Politécnico**

**José Antonio Echeverría**

cujae

**Facultad de Ingeniería Informática**

# PATRONES PARA INCORPORAR PROACTIVIDAD EN SISTEMAS INFORMÁTICOS

**Tesis presentada en opción al grado científico de**
**Doctor en Ciencias Técnicas**

**MAILYN MORENO ESPINO**

**La Habana, Cuba**
**2013**

**Instituto Superior Politécnico**

**José Antonio Echeverría**

cujae

**Facultad de Ingeniería Informática**

# PATRONES PARA INCORPORAR PROACTIVIDAD EN SISTEMAS INFORMÁTICOS

**Tesis presentada en opción al grado científico de**

**Doctor en Ciencias Técnicas**

**Autor:**

**Prof. Auxiliar, Ing. Mailyn Moreno Espino, MSc.**
Facultad de Ingeniería Informática.
Instituto Superior Politécnico José Antonio Echeverría.
La Habana, Cuba
my@ceis.cujae.edu.cu

**Tutores:**

**Prof. Titular, Dr. Ing. Alejandro Rosete Suárez.**
Facultad de Ingeniería Informática.
Instituto Superior Politécnico José Antonio Echeverría.
La Habana, Cuba.
rosete@ceis.cujae.edu.cu

**Prof. Titular, Dra. Ing. Martha Dunia Delgado Dapena.**
Facultad de Ingeniería Informática.
Instituto Superior Politécnico José Antonio Echeverría.
La Habana, Cuba.
marta@ceis.cujae.edu.cu

**Catedrático, Dr. Juan Pavón Mestras.**
Departamento de Ingeniería del Software e Inteligencia Artificial.
Universidad Complutense de Madrid.
Madrid, España.
jpavon@fdi.ucm.es

**La Habana, Cuba**

**2013**

## *Agradecimientos*

A lo largo de este trabajo son muchas las personas que me han dado la mano para ayudar, para darme ánimos y no dejarme caer. Quisiera agradecerles por su compresión y por su ayuda, espero sepan comprender si olvido algún nombre, son las 4 de la mañana y llevo más de 6 meses sin dormir tranquila ;-). Le doy mis agradecimientos a los más cercanos:

A mis padres, por su apoyo eterno, por formarme y amarme. Papi sé que este es un sueño hecho realidad, cuanto quisiera que pudieras verlo y pudieras darme un abrazo. Los amo.

A mis hijas, Lia y Nayma, que han aguantado mi falta, mi mal humor, mi cansancio, su amor es lo más importante que tengo en este mundo.

A mi mana Yoly, por la ayuda, por su amor, por ser una segunda madre para mí y para Lia.

A Alejandro Rosete, sé que ha sido duro para ti ser mi tutor, ha sido una carga muy pesada, agradezco tus azotes, tu ánimo y tu cariño. Sin tu apoyo no hubiera llegado al final.

A Neni y Leo por todo y mucho más.

A Pepin, un beso grande.

A Dany por su ayuda con el inglés.

A Yali, Rodo y Laurita por ser tan buenos amigos.

A Pedro Angel, mi hermano negro, siempre te amaré.

A Marta por sus buenos consejos, por sus despistes ;-) y la ayuda incondicional.

A Juan Pavón, gracias por acogerme sin siquiera conocerme, por tus conocimientos, por tu apoyo, por tu sonrisa cordial, por cada ayuda.

A Raisa y Felix por su revisión exhaustiva como oponentes de Predefensa.

A Sonia, Exiquio, Carli, Anaisa, por sus notas, sus señalamientos, por su ayuda.

A Garay, usted es una persona excepcional, su amistad es para mí un gran tesoro, muchas gracias, lo quiero.

Mayita, en verdad te agradezco, no la tesis, sino mucho más, siempre me sentiré en deuda por cada consejo, por cada ayuda, por cada cariño.

Alfre, hermano mío, siempre cerca en las buenas y en las malas, nadie aguanta mi mal humor como tú, te quiero mucho.

A Alain, por tu apoyo, por su minuciosa revisión, por estar siempre cerca, te quiero.

A Amalia, hermana del alma.

A Sepul, por sus buenos consejos, sus revisiones y contactos ;-)

A los Doctores de la facultad de Informática por sus recomendaciones y señalamientos.

A Cornelio por ayudarme sin siquiera conocerme.

A los compañeros del grupo GRASIA, todo el agradecimiento de mundo, por no hacerme sentir sola y darme sus puntos de vista.

A Rubén por su Teoría de la Actividad.

A Javier, Susana y Ana, por ayudarme desde el primer día que llegué a Madrid sola y desolada.

A Karen, por su amistad y ayuda.

A mi familia española por acogerme en su seno.

A Alternán, sin ti este trabajo no sería posible y lo sabes, te estoy eternamente agradecida por acogerme en el equipo del Observatorio y por confiar en mí.

A Yahima por desandar conmigo el mundo de i* e Ingenias, por su amistad y ayuda, contigo tengo una deuda eterna de gratitud.

A los nenes de PAM por dejarme experimentar con ellos mis teorías.

A Isis por su amistad y su ayuda, por su disco duro, por su risa.

A los niños de Rosete: Jenny, Diana, Daymí, Taymí, David, Ingrita, Bardají

A mis compañeros del Departamento de Inteligencia Artificial, gracias a todos.

A la Chichi y a Guevara.

A los compañeros del CITI por estar pendientes, gracias Lisbán.

A Omar, por estar siempre presto ayudar, por tu No te rindas.

A mis amados estudiantes, que me han dado la oportunidad de guiarlos, dejarme ser su amiga y extraer lo mejor de ellos:

- Hermes, hermano mío
- Kinane, Perla, Leandro Z…, Leandro S…, Harold, Ailín
- Yanelis y Dorys
- Ela, eres la mejor y más llorona
- Yasser, tu simulación, tu transformador, tu mal carácter y tus risas, un besote
- Dainiel, tus locuras, tu cariño, tu amistad
- Taysir, Alexis, Branly, Nelson, Gabriel
- Osmel, eres un caso, al fin llegaste a la meta
- Ale Alonso, tus patrones son mi salvación y por fin la proactividad?
- Lisy, Olver, Notario, Mohamed, Gaby, Jordán, Ariel, Alfonso

A Vilma por su amistad y cariño.

A Sachy un gran beso.

A Aidita, Adita y Ary por estar siempre pendientes de mí.

A Amarilis por su ayuda.

A todos mis profes.

A todos MUCHAS GRACIAS

*A mis hijas*

*A mis padres, a ti donde quiera que estés, Te Amo*

*A mis hermanos, a Mana*

*A Alejandro*

*Al amor, que te hace volar, porque…*

*con tantos palos que te dio la vida*

*y aún no te cansas de decir Te quiero*

## *Síntesis*

Tomar iniciativa para mejorar las circunstancias que se tienen o crear nuevas es tener un comportamiento proactivo. La proactividad es una propiedad que puede beneficiar a los sistemas informáticos. El comportamiento proactivo permite que se le deleguen metas al software y este trabaje para cumplirlas, cuando tenga las condiciones para hacerlo. Un software con comportamiento proactivo puede aportar beneficios para el usuario final que lo utiliza. Las metodologías para el desarrollo de software orientadas a agentes y orientadas a objetos no presentan pasos para detectar la proactividad desde el modelo de los requisitos.

El marco de trabajo i* está orientado a metas y sigue el enfoque del modelado social para la captura de requisitos en software de disímiles perspectivas. A pesar de su fuerte concepción de los actores estratégicos y sus relaciones para cumplimentar metas, no tiene de forma explícita pasos para detectar requisitos proactivos.

El aporte principal de esta tesis es desarrollar patrones que permitan detectar y delegar la proactividad, así como implementar un comportamiento proactivo a partir de la observación periódica de un ambiente.

Como resultado se formularon cuatro patrones para detectar e incorporar proactividad en los sistemas informáticos. Además, se desarrolló un estudio de casos que permitió valorar la generalización de los patrones propuestos. Los dos casos escogidos son muy diferentes en su concepción, uno es la construcción de un observatorio tecnológico y el otro el desarrollo de metaheurísticas proactivas.

# *Tabla de Contenidos*

# *Índice de Figuras*

# *Índice de Tablas*

# *Introducción*

Según Wooldridge [172] la historia de la computación ha estado marcada por cinco tendencias fundamentales que han guiado su desarrollo: la ubicuidad, la interconexión, la delegación, la inteligencia y la orientación a humano. La ubicuidad está dada por la reducción de los costos de la computación, lo que permite encontrar un dispositivo computarizado en casi cualquier lugar, desde la telefonía móvil hasta las lavadoras inteligentes. La interconexión se puede apreciar en que los dispositivos computarizados no se conciben sin estar conectados a una red, por ejemplo Internet, sin embargo hace dos décadas los sistemas de computadoras eran entidades generalmente aisladas. Actualmente se desarrollan sistemas cada vez más complejos y sofisticados. La delegación de tareas complejas a sistemas de cómputos ha permitido desarrollar trabajos tan difíciles y extremos como la aeronáutica y el control de termonucleares. La delegación de tareas viene acompañada por la necesidad de aumentar la inteligencia de las computadoras, no basta con delegar, se deben hacer computadoras más inteligentes, intentando simular la inteligencia de un humano. Este desarrollo de los sistemas computarizados no sería posible si el nivel de abstracción para el desarrollo de software no hubiera ido en aumento. En la actualidad no es necesario ya programar en código ensamblador, se pueden utilizar lenguajes de alto nivel, que hacen posible desarrollar sistemas capaces de responder a las necesidades de la sociedad altamente tecnológica de hoy.

Todas las tendencias que han marcado la computación [172] han provocado el surgimiento de un paradigma que ha permitido aumentar el nivel de abstracción donde se encapsulan datos, métodos y decisión, este es el paradigma de los agentes [40, 92, 150, 165, 172, 173]. Según la Fundación para los Agentes Inteligente Físicos (FIPA, *Foundation for Intelligent Phisical Agents*) [51] un agente es una entidad de software con un grupo de propiedades entre las que se destacan ser capaz de actuar en un ambiente, comunicarse con otros agentes, estar condicionado por un conjunto de metas, manejar recursos propios, ser capaz de percibir su ambiente y tomar de él una representación parcial, etc. De forma

general, varios autores reconocen en los agentes diversas propiedades: autonomía, reactividad, proactividad y tener habilidad social [40, 92, 165, 172].

En la sociedad actual la proactividad es una de las características que tienen los humanos para cambiar su entorno, es un rasgo de inteligencia. Tomando iniciativas, el hombre es capaz de generar ideas creativas [57]. En muchas áreas la proactividad es una propiedad deseada y que tiene buenos resultados. La proactividad se trata de tomar el control para hacer que las cosas sucedan, no esperar a que ocurran eventualmente [76].

La proactividad es una de las características más distintivas de los agentes [129, 172]. La proactividad es un comportamiento dirigido por metas, el agente trabaja para alcanzar una meta. El comportamiento proactivo permite que se le delegue las metas al software y este trabaje para cumplirlas. Un software con comportamiento proactivo en áreas como la gestión, la toma de decisiones [104, 159], los ambientes asistidos [106, 110] y los sistemas en tiempo real [69], es un beneficio para el usuario final que lo utiliza. En otros campos de la inteligencia artificial como son las metaheurísticas se han utilizado los agentes [74, 102, 138] para la cooperación y la competencia, pero no para dotar a las metaheurísticas de comportamientos proactivos.

Hasta ahora en la Informática el término de proactividad ha sido tratado mayormente por los trabajos relacionados con el paradigma de agentes [28, 165, 172]. El surgimiento de este nuevo paradigma impone retos en las técnicas y modelos de la Ingeniería de Software que estaban estandarizadas [83, 90]. Antes de que los agentes tuvieran la fuerza que tienen hoy, el paradigma orientado a objetos ya era un paradigma fuerte y estable, y en el día de hoy lo sigue siendo.

El paradigma de orientación a objetos se utiliza mucho actualmente, debido a las facilidades que brinda para la reutilización del código [13]. En el desarrollo del software orientado a objetos ha tomado auge la utilización del Proceso Unificado de Desarrollo (RUP, *Rational Unified Process*) [90], que constituye una propuesta de proceso para el desarrollo de software orientado a objetos que utiliza el Lenguaje Unificado de Modelado (UML*, Unified Modelling Language*) [55, 143]. El

auge en el uso de RUP y UML hace de la orientación a objetos un paradigma robusto y maduro.

Sin embargo, los objetos no son autoiniciables, no tienen la iniciativa de hacer algo sin una petición u orden [24]. RUP es un proceso de desarrollo dirigido por casos de uso. Según [90] “Un caso de uso especifica una secuencia de acciones, incluyendo variantes, que el sistema puede llevar a cabo, y que producen un resultado observable de valor para un actor concreto”. La comunicación para iniciar un caso de uso es a través de un mensaje o petición de un actor. Este comportamiento es intrínseco en la orientación a objetos porque los objetos trabajan para dar respuesta a un mensaje [24]. Al estar enfocados al desarrollo de software orientado a objetos, los procesos como RUP no conciben una manera de tratar con la proactividad, no la modelan y no lo tienen en cuenta como un requisito posible del software.

El paradigma de agentes ha evolucionado hasta llegar a tener metodologías y lenguajes capaces de sustentar las características más comunes que se esperan en un software de agentes [16, 28, 84]. Entre las más conocidas están: Tropos [6, 23, 115], Ingenias [66, 72, 137], GAIA [178, 179], PASSI [82], MaSE [37, 38] y Prometheus [66, 132]. Junto con estas metodologías han surgido extensiones de UML, tales como AUML (*Agent UML*) [12], para poder representar las características de los agentes que no exhiben los objetos. Ingenias es una de las metodologías orientada a agentes más completa, ya que tiene todo un ciclo de desarrollo que permite llegar a un producto final de calidad [84, 117, 118, 119]. Tropos es una metodología robusta en la captura de requisitos ya que adopta el marco de trabajo i* [84, 117, 118, 119]. También han surgido plataformas de desarrollo de agentes, entre las que destaca JADE (*Java Agent DEvelopment Framework*) [14, 15] que permite desarrollar un sistema con agentes siguiendo los estándares de FIPA [50, 51, 52].

No obstante a la cantidad de metodologías surgidas a raíz del avance del paradigma de agentes, todas se enfocan a tratar problemas como la comunicación entre los agentes, la organización y los protocolos de comunicación. No se

enfocan en cómo modelar la proactividad, cómo descubrirla desde los requisitos tempranos de un software, y así aumentar las prestaciones de cara al usuario.

La proactividad, tanto en los humanos como en los agentes, parte de una representación de las metas a cumplir [92]. Se puede llegar a ser proactivo trazando metas y tomando iniciativas siguiendo estas metas.

El modelado social es un nuevo enfoque en el reto que presupone la captura de requisitos de un software [177]. 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 [89]. Un elemento a destacar es que la orientación a agentes aprovecha los puntos fuertes de la orientación a metas [177].

El marco de trabajo y lenguaje i* se utiliza para la captura de requisitos siguiendo el enfoque de modelado social [176]. i* reconoce la primacía de los actores sociales, los cuales son vistos como intencionales, es decir, tienen metas, creencias, habilidades y compromisos. El análisis se enfoca en satisfacer la forma en que se capturan las metas de los distintos actores, dada alguna configuración de las relaciones entre los actores humanos y el sistema. La reconfiguración de estas relaciones puede ayudar a plasmar los intereses estratégicos de los actores [175]. Aunque con i* se puede expresar la relación entre los actores estratégicos del sistema a desarrollar, no se conoce ningún método que emplee i* para detectar requisitos proactivos que puedan beneficiar a un software [89].

En este contexto son importantes los patrones. Christopher Alexander [5] 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”. Se han desarrollado patrones para muchos contextos, sobre todo en la orientación a objetos. Los patrones más conocidos son los de diseño [63], los que están específicamente enfocados a la orientación a

objetos. También se ha trabajado en patrones de implementación [13]. Estos permiten que el trabajo de los programadores sea más efectivo a medida que invierten menos tiempo en partes repetitivas de su trabajo y le dediquen más tiempo a resolver problemas verdaderamente únicos.

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 [9], para la comunicación entre agentes [131, 148] y para la organización de un sistema de agentes [29, 145, 179]. 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. La mayoría de estos patrones se enfocan en el diseño y no en la implementación. En esta dirección, no se conoce sobre ningún trabajo enfocado en simplificar el trabajo con JADE [14], encapsulando la solución de problemas comunes en la construcción de un sistema multi-agente (SMA). Particularmente, estos dos aspectos (la proactividad, y la simplificación del trabajo con JADE) son dos problemas comunes en muchas soluciones basadas en SMA, para las cuales no se conocen que existan patrones definidos.

A partir de lo anterior se puede identificar el siguiente **problema de investigación:**

Ausencia de patrones para incorporar proactividad en el desarrollo de un sistema informático.

Para enfrentar este problema, se definió como **objeto de la investigación**: las metodologías orientadas a objetos y orientadas a agentes, especialmente el modelado social y los patrones, y como **campo de acción**: el lenguaje de modelado i* y los patrones de requisitos y de implementación.

Para responder al problema de investigación, se definió el siguiente **objetivo general**:

Formular patrones de requisitos y de implementación para incorporar comportamientos proactivos en los sistemas informáticos.

A partir del análisis del objetivo general se derivaron los siguientes **objetivos específicos**:

1. Formular patrones de requisitos que permitan identificar comportamientos proactivos a partir de los requisitos modelados en i*.
2. Formular patrones de implementación que permitan incorporar proactividad.
3. Evaluar los patrones a través del estudio de casos.

Se formuló la siguiente **hipótesis** de investigación:

El uso de patrones de requisitos basados en los modelos de i* y de patrones de implementación posibilitan incorporar comportamientos proactivos en sistemas informáticos.

Para lograr los objetivos trazados y demostrar la hipótesis establecida se definieron las siguientes **tareas**:

1. Estudiar el estado del arte de la ingeniería de software orientada a objetos, las metodologías orientadas a agente, la proactividad y el modelado social en particular.
2. Desarrollar patrones de requisitos que, a partir del modelo de los requisitos tempranos de i*, permitan delegar metas proactivas al software a desarrollarse, reflejando esto en el modelo de los requisitos tardíos de i*.
3. Desarrollar un patrón de implementación que simplifique la configuración de JADE, brinde soporte a un patrón de observación periódica y permita hibridar objetos y agentes.
4. Desarrollar un patrón de implementación que permita incorporar proactividad a partir de la observación periódica.
5. Desarrollar un estudio de caso en dos contextos de análisis: el desarrollo de un Observatorio Tecnológico y la obtención de requisitos proactivos en sistemas complejos como las metaheurísticas.

Entre los **métodos de trabajo científico** utilizados en la investigación se destacan los que se mencionan a continuación. Además, se brinda una breve descripción de los fines para los cuales fueron utilizados.

- Métodos generales: el método hipotético-deductivo para la elaboración de la hipótesis central de la investigación y para proponer nuevas líneas de trabajo a partir de los resultados parciales. El método sistémico para lograr que los elementos que forman parte de los patrones sean un todo que funcione de manera armónica. El método histórico-lógico para el estudio de los trabajos anteriores y extraer aspectos positivos de ellos, utilizando éstos como punto de referencia y comparación de los resultados alcanzados.
- Métodos lógicos: el método analítico-sintético al descomponer el problema de investigación en elementos por separado y profundizar en el estudio de cada uno de ellos, para luego sintetizarlos en la propuesta de solución.
- Métodos empíricos: el método coloquial para la presentación y discusión de los resultados en sesiones científicas. El método experimental, utilizando el estudio de casos para comprobar los métodos propuestos, así como en las pruebas estadísticas que se aplicaron. El análisis documental, empleado en el trabajo de revisión bibliográfica.

La **novedad científica** del trabajo radica en:

-dos patrones de requisitos para detectar posibilidades de proactividad en sistemas informáticos;

-dos patrones de implementación para simplificar el trabajo con JADE e incorporar características proactivas a partir de una observación periódica en sistemas informáticos;

-la incorporación de proactividad en cuatro metaheurísticas para ajustar parámetros y ajustar las vecindades de forma proactiva.

El **valor práctico** del trabajo es el siguiente:

-utilidad de los patrones propuestos para desarrollar sistemas con comportamientos proactivos que se han empleado en el desarrollo de un

observatorio tecnológico y en otras tareas de interés operativo para órganos de la defensa nacional;

-propuestas de nuevas metaheurísticas proactivas que son aplicables en muchos contextos.

Los principales resultados que se han obtenido, vinculados directamente con la presente tesis, han sido presentados en eventos y publicaciones, tanto nacionales como internacionales. Al desarrollo de esta investigación que se resume en este documento han tributado dos tesis de maestría [81, 119], dos tesis de diplomado [64, 80] y 16 tesis de diploma. Para ver los detalles de la producción científica del autor sobre el tema de la tesis se puede consultar el anexo 1.

La tesis quedó estructurada en tres capítulos: Fundamentos Teóricos, Patrones de requisitos e implementación para incorporar proactividad y Estudio de de Casos: desarrollo de sistemas informáticos proactivos.

En el Capítulo 1, Fundamentos Teóricos, se hace un análisis crítico de las metodologías de desarrollo de software, tanto orientadas a agentes como orientadas a objetos, en cuanto a su forma de tratar la proactividad. Además, se desarrolla un análisis del lenguaje i* dirigido por metas y un estudio crítico de los patrones. En el Capítulo 2, Patrones de requisitos e implementación para incorporar proactividad, se desarrollan cuatro patrones para incorporar proactividad en los sistemas informáticos. Se presentan además actividades de pruebas que se relacionan con los patrones propuestos. En el Capítulo 3, Estudio de de Casos: desarrollo de sistemas informáticos proactivos, se presentan dos casos donde se aplican los patrones propuestos, siguiendo el estudio de casos como método de investigación y evaluación.

# *Capítulo 1*

# *Fundamentos Teóricos*

# *Capítulo 1: Fundamentos Teóricos*

## 1.1 Introducción

En este capítulo se profundiza en el término de proactividad. Se exponen las limitaciones de la orientación a objetos para tratar con la proactividad y los retos que llevaron al surgimiento de los agentes. Se estudia la definición de agente, las principales metodologías orientadas a agentes y sus limitaciones en cuanto a la carencia de pasos explícitos para encontrar requisitos proactivos. Se desarrolla un estudio del modelado social como un enfoque para capturar requisitos y más específicamente del lenguaje i* por estar dirigido a metas. Se hace una breve presentación de JADE como plataforma de desarrollo de sistemas multi-agente. Se realiza un estudio de los patrones como formas para resolver un problema común. Se comienza por los patrones en la orientación a objetos y luego se tratan los de la orientación a agentes. Por último, se expone el Modelo-V, que proporciona una forma de entrelazar las fases del ciclo de vida del desarrollo de un software con las actividades de pruebas para cada fase.

## 1.2 Proactividad

El término proactividad es utilizado en la Psicología desde los años 40 del pasado siglo. En 1946 Viktor Frankl en [57] plantea que:

“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, haciendo prevalecer la libertad de elección sobre las circunstancias de la vida.”

En el área de la administración y dirección, el comportamiento proactivo ha sido conceptualizado de muchas formas. Una de las definiciones más utilizadas es: “el comportamiento proactivo es tomar iniciativa para mejorar las circunstancias actuales o crear nuevas; esto involucra bastantes cambios en el *statu quo* de la adaptación pasiva de las condiciones presentes” [31].

La proactividad se trata de tomar el control para hacer que las cosas sucedan, no esperar que ocurran eventualmente [76]. Según [133] la proactividad tiene tres

atributos fundamentales: comienza por sí misma, está orientada al cambio y se enfoca en el futuro. La proactividad se divide en tres niveles: individual, de equipo y organizacional [134].

La proactividad en el 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 [92].

Existen trabajos que demuestran lo conveniente que puede ser la proactividad para sistemas informáticos en contextos diversos [10, 43, 62, 96, 97, 98, 99, 101, 103, 105, 108], sin embargo, en ninguno de ellos se expresa de manera clara qué método se utilizó para llegar a la proactividad.

## 1.3 Desarrollo de software en la orientación a objetos

La Ingeniería de Software ha evolucionado por diferentes etapas para llegar a lo que existe hoy en día [140, 156, 171]. Por ejemplo, pasó por el enfoque estructurado y luego llegó el orientado a objetos. Este último fue un cambio en la forma de pensar acerca del proceso de descomposición de problemas [24]. Un objeto encapsula estados (valores de datos) y comportamientos (operaciones). En la programación orientada a objetos la acción se inicia mediante la transmisión de un mensaje al objeto. Un objeto exhibirá su comportamiento mediante la invocación de un método como respuesta a un mensaje [24].

En nuestros días hay una tendencia elevada hacia el enfoque orientado a objetos en los sistemas que se construyen [90, 155], debido a las facilidades que brinda para la reutilización del código entre otros aspectos [13].

En el desarrollo del software orientado a objetos ha tomado auge la utilización del Proceso Unificado de Desarrollo (*RUP*, *Rational Unified Process*) [90], que es una propuesta de proceso para el desarrollo de software orientado a objetos que utiliza el Lenguaje Unificado de Modelado (*UML, Unified Modelling Language*) [55, 143].

*RUP* es un proceso de desarrollo dirigido por casos de uso [90]. Los casos de uso no son autoiniciables, no tienen la iniciativa de hacer algo sin una petición u orden.

Los objetos trabajan para dar respuesta a un mensaje y están guidados por eventos [24].

En la captura de requisitos propuesta por *RUP* no se representan explícitamente las intenciones de los actores estratégicos. La captura de requisitos está basada principalmente en los casos de uso y los actores que lo inicializan. Por otro lado, los casos de uso se pueden ver de una forma más detallada en un diagrama de actividades, que permite expresar qué actividad le corresponde a un actor o al sistema, pero no permite ver la dependencia y las intenciones de los actores [176].

Con la introducción del Modelo del Negocio se produjo una evolución en RUP [90]. Esta se debió a que el Modelo del Negocio permitió modelar los problemas de una organización y la necesidad de automatización de algunas de sus partes. En el Modelo del Negocio las dependencias entre los actores y la intencionalidad no se muestra en un solo diagrama [90, 177]. El modelo tampoco permite ver las metas de los actores, consta no obstante de un artefacto, no en el lenguaje UML, que muestra el desglose de las metas de la organización pero que no especifica los actores responsables [90]. *RUP* no proporciona una guía para encontrar y modelar comportamientos dirigidos a metas, tratar explícitamente el nivel intencional de los actores estratégicos y sus dependencias. Es decir no maneja la proactividad como principio [83, 156].

La ingeniería de software orientada a objetos planteó la necesidad de nuevas técnicas para descomponer (dividir en pedazos más pequeños que puedan tratarse independientemente), abstraer (posibilidad de modelar concentrándose en determinados aspectos y obviando otros detalles de menor importancia) y organizar jerárquicamente (posibilidad de identificar organizaciones, gestionar las relaciones entre los componentes de la misma solución, que incluso permitan su tratamiento de grupo como un todo, según convenga, y ver cómo lograr que entre todos se haga la tarea) [92]. En esta misma línea de desarrollo de software para ambientes distribuidos tomó fuerzas el paradigma de agentes [92, 150].

Construir software que resuelva problemas de negocios actuales no es una tarea fácil. Al incrementarse las aplicaciones complejas, demandadas por diversos tipos

de negocios y competir con ventaja en el mercado, las tecnologías orientadas a objetos pueden ser complementadas por las tecnologías orientadas a agentes [84].

## 1.4 Agentes Inteligentes

Las tendencias que han caracterizado la historia de la computación han dado lugar a la aparición de un nuevo paradigma en la ciencia de la computación: los agentes [172]. Con el fin de interactuar con éxito, los agentes requieren la capacidad de cooperar, coordinar y negociar con los demás, de forma similar a como lo realizan las personas en la vida cotidiana.

Aunque no hay total coincidencia en cuanto a qué es un agente, un intento de unificar los esfuerzos para el desarrollo de esta tecnología puede encontrarse en FIPA [49]. FIPA define al agente como una entidad de software con un grupo de propiedades, entre las que se destacan: ser capaz de actuar en un ambiente, comunicarse directamente con otros agentes, estar condicionado por un conjunto de tendencias u objetivos, manejar recursos propios, ser capaz de percibir su ambiente y tomar de él una representación parcial, ser una entidad que posee habilidades y ofrece servicios, que puede reproducirse, entre otras [51].

De forma general, varios autores reconocen en los agentes diversas propiedades, entre las que se destacan el ser autónomos, reactivos, proactivos y tener habilidad social [107, 151, 172, 173]. Con ligeras modificaciones de enfoques, otros autores también reconocen estas propiedades [40, 53, 56, 127, 165]. Algunos ven a un agente como un objeto activo con iniciativa [135].

De forma general la autora de este trabajo considera que las anteriores definiciones son válidas, con distintos grados de amplitud y reflejando aspectos diferentes, aunque ninguna entra en contradicción con otra. Se puede decir que las características fundamentales de los agentes son: autonomía, reactividad, proactividad y habilidad social, que se exponen a continuación. [68, 84, 151, 172]:

**Autonomía:** un agente opera sin la intervención directa de humanos y tiene cierto control sobre sus acciones y su estado interno.

**Reactividad:** los agentes perciben su entorno y responden en un tiempo razonable a los cambios que ocurren en él. El agente puede estar en estado pasivo la mayor parte del tiempo y despertar en el momento que detecte ciertos cambios.

**Proactividad:** los agentes no solo responden a cambios, sino que pueden tener un comportamiento con iniciativa propia dirigido hacia una meta.

**Habilidad social:** los agentes tienen la capacidad de interactuar con otros agentes mediante algún mecanismo de comunicación. Esto les permite lograr objetivos que por sí solos no pueden lograr. [58].[144].

En la Figura 1 se puede ver una visión esquemática de un agente [95].

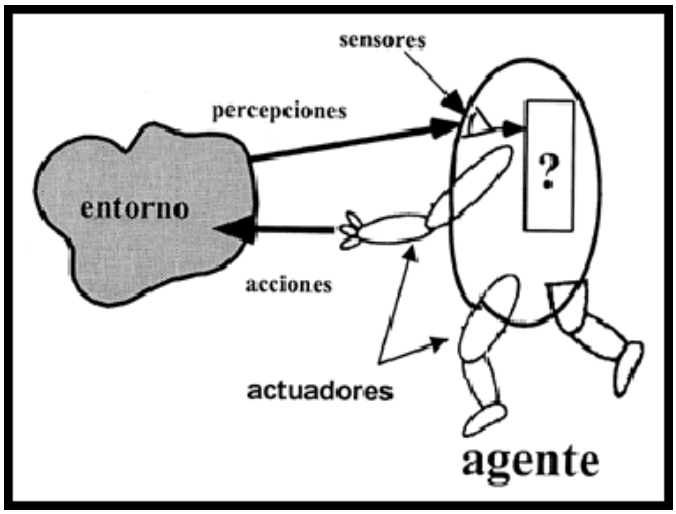



Figura 1: Visión esquemática de un agente.

Existen varias teorías para modelar a los agentes. El modelo BDI es el más extendido, sobre todo para modelar los agentes racionales. El modelo fue propuesto por Rao y Goergeff [67] y toma su nombre de tres aspectos relevantes: las creencias (*belief*), los deseos (*desire*) y las intenciones (*intention*).

Los modelos BDI tienen un origen común al término de Programación Orientada a Agentes (AOP) dado por Yoav Shoham en 1993 [150]. Si se establece un paralelo entre la orientación a objetos y la orientación a agentes se puede relacionar a las creencias con los atributos pasivos (datos) y los deseos con los atributos activos

(métodos). Sin embargo, las intenciones no se manejan de manera explícita en la orientación a objetos porque no hay metas en los objetos.

BDI es el modelo teórico más conocido, aunque en la práctica puede no adoptarse un modelo tan teórico para desarrollar un sistema práctico, sino que se asigna a cada agente un conjunto de propiedades que varían su nombre según la metodología que se utilice para desarrollar el agente. Una metodología lo ve como permisos, actividades, tareas y reglas de interacción [113]; otra lo ve como metas y normativas [4]. En la Figura 2 se muestra una visión esquemática de la arquitectura BDI.

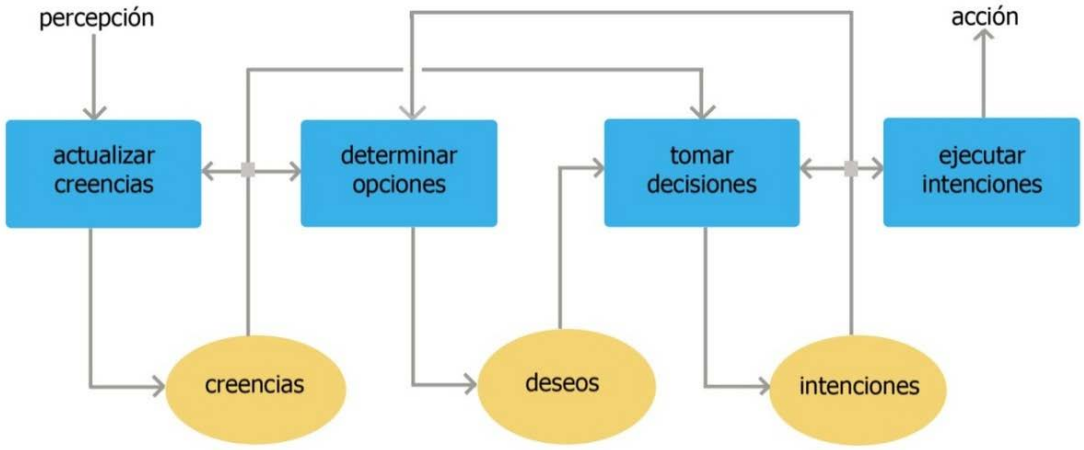



Figura 2: Arquitectura BDI.

## 1.5 Desarrollo de software en la orientación a agentes

Una de las propiedades más importantes en los agentes es su habilidad social [93]. Los agentes sirven para abstraer en ellos elementos de alto nivel (subsistemas), que involucran las intenciones, de forma similar a como las organizaciones humanas estructuran una tarea. Por esa razón, puede ser importante, desde un inicio, definir las reglas de la organización de los agentes y existen metodologías que se centran en esto [114, 172, 178].

Debido a la complejidad de muchos sistemas, no tiene sentido intentar predecir todas las interacciones posibles, por lo que a los agentes se les prepara para tener acciones en el futuro, iniciar acciones y decidir qué hacer [4]. De esta forma se reduce el acoplamiento, ya que se prepara a los componentes para interacciones

imprevistas y a generar soluciones cooperadas. Estas interacciones ocurren en un nivel más semántico. Esto ayuda a reducir la existencia de errores inesperados de sintaxis y de ejecución. De cierta manera pueden verse como una evolución en el mismo sentido que el manejo de excepciones.

Al tener estas propiedades y estar activos siempre, la organización de agentes está preparada para que el comportamiento emerja de abajo hacia arriba. Así, es más fácil cambiar la estructura organizativa cuando proceda, o ampliar su uso. Por tanto, promueve una amplia reusabilidad, incluso permitiendo manejar la posibilidad de crecer sistemas abiertos (donde entran o salen agentes dinámicamente). Predecir el comportamiento del sistema es un reto [91, 165], a pesar de los intentos por vías formales usando axiomas y demostraciones [165].

Por lo general, los agentes no son desarrollados como elementos independientes, sino como entidades que constituyen un sistema. A este sistema se le denomina multi-agente. En este caso, pueden interactuar entre ellos. Las interacciones más habituales (como son informar o consultar a otros agentes) permiten a los agentes «hablar» entre ellos, tener en cuenta lo que realiza cada uno y razonar acerca del papel jugado por los diferentes agentes que constituyen el sistema. La comunicación entre agentes se realiza por medio de un lenguaje de comunicación de agentes (uno de los más conocidos es FIPA-ACL: *Agent Communication Language* [50, 68]).

Los sistemas multi-agente (SMA, o MAS por sus siglas en inglés: *Multi-Agent Systems*) pueden entenderse como grupos de agentes que interactúan entre sí para conseguir metas comunes. Una definición de un SMA se puede encontrar en [46], donde se dice que este reúne los siguientes elementos:

- Un ambiente.
- Un conjunto de objetos integrados con el ambiente. Dichos objetos son pasivos, pueden ser creados, modificados y destruidos por los agentes.
- Un conjunto de agentes que representan las entidades activas del sistema.

- Un conjunto de operaciones que hacen posible el trabajo de los agentes sobre los objetos.

Según esta definición, la influencia de unos agentes sobre otros viene dada no sólo por la comunicación explícita, sino también por la actuación sobre el entorno. Este hecho aumenta la complejidad del desarrollo del SMA enormemente, ya que obliga a estudiar el entorno en detalle para detectar cuáles acciones realizadas por un agente pueden afectar a otro. La complejidad del ambiente viene dada por muchos factores como son su carácter discreto o continuo, su dinamismo, o determinismo, y esta complejidad implicará retos mayores para los agentes [144, 172].

En el desarrollo de SMA existen dos filosofías [136]. La primera ve el SMA como el resultado de utilizar un lenguaje de especificación de agentes. Para generar SMA de esta manera, se parte de principios basados en modelos operacionales y modelos formales. La segunda estudia el SMA como un sistema software que hay que construir. El desarrollo no parte de cero, sino que utiliza plataformas de desarrollo de agentes que proporcionan servicios básicos de comunicación, gestión de agentes y una arquitectura de agente [136]. Más adelante, en este trabajo se aborda una de las plataformas más utilizada para el desarrollo de un SMA. En cualquiera de los dos casos, y sobre todo cuando el sistema a desarrollar es grande, se necesitan metodologías que estructuren el desarrollo de acuerdo con las prácticas de la Ingeniería de Software Orientada a Agentes [28, 83].

El desarrollo de la Ingeniería de Software Orientada a Agentes ha traído consigo el surgimiento de muchas metodologías orientadas a agentes. Entre las más destacadas están: Prometheus [66, 132], Passi [82], Tropos [6, 23, 115] e Ingenias [66, 72, 137]. En la Tabla 1 se pueden ver los flujos del ciclo de vida que cubren estas metodologías [84, 118, 119]. En la Tabla 1 se utiliza como nomenclatura la S para decir que sí cubre la etapa, P para representar que propone la etapa pero no forma parte de la metodología y N para representar que no cubre la etapa.

Tabla 1: Comparación de metodologías: Etapas del ciclo de vida.

| | Tropos | INGENIAS | PASSI | Prometheus |
|---|---|---|---|---|
| Requisitos | S | P | P | N |
| Análisis | S | S | S | S |
| Diseño | S | S | S | S |
| Implementación | S | S | S | N |
| Prueba | P | P | N | N |

Como se puede observar, Tropos es de las metodología más completas, concibe el desarrollo del SMA desde los requisitos hasta su fase de prueba [127]. Para la captura de requisitos, Tropos utiliza el lenguaje i* [77, 176], que sigue la filosofía del modelado social [177]. Sobre este lenguaje y sus ventajas en la captura de requisitos se aborda en la próxima sección por lo relevante que es para este trabajo. Tropos, a pesar de utilizar el modelado social, no explota las posibilidades de encontrar desde los requisitos tempranos las relaciones o dependencias que se puedan delegar en el software y de esta forma tener un comportamiento proactivo. Ingenias, por otro lado, es una metodología robusta en cuanto al desarrollo de un SMA, desde su análisis hasta su implementación [73, 84, 87]. Para los requisitos, Ingenias propone seguir los pasos de *RUP* o utilizar la Teoría de la Actividad [59, 60, 61], en ambos casos la fase no está sustentada por su herramienta *CASE* [72, 136]. Como se planteó anteriormente, *RUP* no trata la proactividad por lo que Ingenias tiene una limitación en cuanto a detectar requisitos proactivos por esta vía. La Teoría de la Actividad tiene un principio similar al modelado social, pero adolece de no constar con pasos para detectar la proactividad. En el caso de PASSI sigue también los principios de *RUP* para la captura de requisitos. Aunque todas estas metodologías tienen una fase de requisitos de una forma más o menos detallada, ninguna muestra de forma explícita los pasos para encontrar comportamientos proactivos desde el modelado de sus requisitos. En el desarrollo del software basado en agentes la proactividad es tratada en fases como el diseño y la implementación y se deja su inclusión a la experticia del arquitecto de software o en respuesta a una petición clara de los clientes.

## 1.6 Modelado Social: Una visión de los requisitos

Desde el inicio del desarrollo de sistemas informáticos existe un gran problema, que es la identificación de los requisitos. Esto se debe a que no es un proceso que pueda ser formulado matemáticamente, por el contrario, es un proceso en el cual los datos son extraídos de las personas y estos pueden variar dependiendo de la persona a quien se esté consultando.

Hoy en día, 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, analizando necesidades, confirmando su viabilidad, negociando una solución razonable, especificando la solución sin ambigüedad, validando la especificación y gestionando los recursos para que se transformen en un sistema operacional [154, 157].

Se le llama captura de requisitos al acto de descubrimiento, 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. Es el proceso de averiguar, normalmente en circunstancias difíciles, lo que se desea construir [90].

La captura de requisitos incluye varias actividades en las cuales la interacción con los clientes tiene una importancia primordial [60, 61]. No sólo se trata de obtener de los clientes conocimiento sobre los objetivos del sistema a construir o su contexto, sino también de que los desarrolladores sean capaces de comunicar la visión que han obtenido de dichos requisitos y ambos grupos puedan negociar sobre ellos si es preciso. Considerar estas actividades, en relación con el manejo de las propiedades de los agentes, resalta la necesidad de usar lenguajes apropiados para representarlas y razonar sobre ellas. Además, estos lenguajes han de ser comprensibles tanto por los clientes como por los desarrolladores, ya

que los requisitos son un artefacto construido conjuntamente por ambos grupos [59].

Las tecnologías de la información pueden ser utilizadas de diversas maneras y tiene un gran potencial para mejorar la vida de las personas. No obstante, el diseño de sistemas que realmente respondan a las necesidades de las personas sigue siendo un desafío. A diario se encuentran sistemas que no hacen lo que se espera de ellos. Gran parte de este problema se debe a que los requisitos del sistema no se capturan de forma rigurosa [176].

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 [177]. 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. El mundo ha cambiado, pocos habrían predicho la forma que en se ha desarrollado la revolución de los sistemas de información, el uso informático forma parte de todo hoy [176].

La evolución de los sistemas de software y de los problemas a los que se enfrentan actualmente las tecnologías de la información impone adoptar un punto de partida diferente para comprender el mundo en el que el sistema de información estará situado. El tradicional análisis de requisitos adopta nuevas posturas, ya que el mundo está compuesto por entidades y actividades que son conocibles y predecibles [77].

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 cuáles acciones tomar, lo cual los hace autónomos. Los actores no existen de forma aislada. Existen en algún entorno compartiendo e interactuando con otros [176].

El modelado social, al enfocarse en etapa temprana de la Ingeniería de Requisitos, se centra en la dimensión social de los sistemas informáticos y su

entorno. Es parte de un método de ingeniería que proporciona técnicas sistemáticas y herramientas que pueden proporcionar un vínculo claro con el resto del proceso de desarrollo del sistema, incluyendo el diseño y la ejecución [176]. 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.

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 [89]. Algo que hay que destacar es que la orientación a agentes aprovecha los puntos fuertes de la orientación a metas [177].

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 [176]. 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 [175].

## 1.7 El lenguaje y marco de trabajo i*

Los modelos orientados a metas se han convertido en una herramienta frecuente en la Ingeniería de Requisitos. Actualmente existen diversos lenguajes para la construcción de modelos orientados a metas, destacándose entre ellos el marco de trabajo i*. El marco de trabajo i* permite expresar, de forma clara y sencilla, las metas de los actores que aparecen en los modelos y las dependencias entre ellos. Además, cuenta con una notación gráfica que permite tener una visión intuitiva y unificada del entorno modelado, mostrando los actores y dependencias. Estas representaciones se realizan mediante el lenguaje conceptual de modelado Telos [125, 175], que conforma la base del lenguaje i* [175]. Este marco de trabajo

permite capturar los requisitos del SMA de una forma muy lógica y coherente, aunque se puede utilizar para modelar sistemas orientados a objetos. Para desarrollar sistemas que verdaderamente encuentren las necesidades reales de una organización se necesita tener una comprensión más profunda de cómo está empotrado el sistema en el ambiente organizativo. Para ello en i* se propone la captura los requisitos en dos fases, Análisis de Requisitos Tempranos y Análisis de Requisitos Tardíos. Además, hace uso de dos modelos, cada uno correspondiente a un nivel diferente de abstracción: el nivel intencional representado por el Modelo de Dependencia Estratégica (*Strategic Dependency Model*, SD) y el nivel racional representado por el Modelo Estratégico de Racionalidad (*Strategic Rational Model*, SR) [175].

i* se ha utilizado con éxito en diferentes proyectos. Un ejemplo es el caso de Eurocontrol, que es la organización que supervisa el espacio aéreo europeo. Se utilizó i* por la necesidad de modelar y especificar sistemas técnicos que deben cooperar [177]. En el proyecto CORA-2 (*Conflict Resolution Assistant*), con i* se especificaron los requisitos de los actores estratégicos de un sistema que provee asistencia computarizada a los controladores del tráfico aéreo para resolver conflictos potenciales entre aeronaves en vuelo. En el desarrollo de este proyecto se encontraron propiedades potenciales, tales como, guías para separar los roles de los humanos en diferentes actores del sistema y para modelar las dependencias de recursos [177]. Se ha utilizado con éxito en sistemas de salud [8] y en sistemas que censan un entorno [75]. También se ha utilizado en temas de seguridad con su herramienta Si*[1], que da soporte de riesgos desde los requisitos [35]. Se han desarrollado estudios empíricos donde se ha obtenido que el modelado de metas en i* es superior a los casos de uso en términos de comprensibilidad [79]. A continuación se presentan las fases de i* y sus conceptos básicos, para consultar con más detalles se puede consultar el anexo 2 y la abundante bibliografía del tema [77, 175, 176, 177].

[1] Si*: Herramienta de seguridad de TROPOS

### 1.8.1 Análisis de requisitos tempranos

Esta etapa consiste en la identificación y análisis de los principales actores estratégicos (*Stakeholders*) del dominio involucrado en el problema y de sus necesidades e intenciones [23, 115].

El análisis de requisitos tempranos permite modelar las relaciones de los actores estratégicos. Normalmente, cuando se intenta entender una organización, la información capturada por las técnicas de modelado no son capaces de expresar las razones (el "por qué") del proceso (las motivaciones e intenciones) [176].

El análisis de requisitos tempranos está estrechamente relacionado con la comprensión de un determinado problema existente en alguna organización. Los actores estratégicos se modelan como actores sociales que se relacionan y dependen unos de otros para cumplimentar determinados objetivos, realizar planes y explotar recursos. Exponiendo claramente estas dependencias es posible plantear el "por qué", además del "qué" y el "cómo" de las funcionalidades del sistema, además de verificar cómo el resultado final corresponde con las necesidades iniciales [177].

### 1.8.2 Análisis de requisitos tardíos

El análisis de requisitos tardíos se centra en el "qué" de las cosas, o sea, en dejar lo más claro posible "qué" debe hacer el nuevo sistema. Se enfoca al sistema dentro del ambiente que opera, junto con las características y las funciones relevantes. El sistema es representado como un actor que tiene varias dependencias con los actores de la organización, estas dependencias definen los requisitos funcionales y no funcionales del sistema, es decir, durante esta fase se describe al sistema a desarrollar como otro actor en su entorno operacional, interactuando con otros actores, creando dependencias y definiendo sus objetivos y planes para alcanzarlo [23, 115].

La diferencia más marcada es que en esta fase el futuro sistema se introduce en el modelo de análisis como uno de los actores del entorno global. A partir del momento en que se define este paso, la labor del Ingeniero de Software se "restringe" a analizar y descomponer los objetivos del sistema. Si bien esta es una

diferencia notable y bien pudiera plantearse como la más importante, no es la única divergencia con los Requisitos Tempranos. Otra disimilitud es que el actor representante del sistema se describe con un nivel superior de autonomía e intencionalidad si se compara con los actores sociales. Además, el análisis de las metas del sistema debe realizarse desde el ámbito del sistema, y de ese modo minimizar las modificaciones a las decisiones tomadas durante la fase de requisitos tempranos [25, 81, 115].

En esta fase debería quedar claro qué metas se delegan al sistema, pero i* en esta fase adolece de pasos que permitan ver las intenciones que pueden ser proactivas y cómo se pueden delegar a pesar de que se cuenta con la información de las metas y de las relaciones entre estas y las tareas.

### 1.8.3 Conceptos básicos de i*

i* maneja conceptos básicos necesarios para modelar los actores y sus relaciones. En la Figura 3 se muestran los estereotipos de algunos conceptos modelados con Taom4E [17], la herramienta *CASE* de Tropos [115].

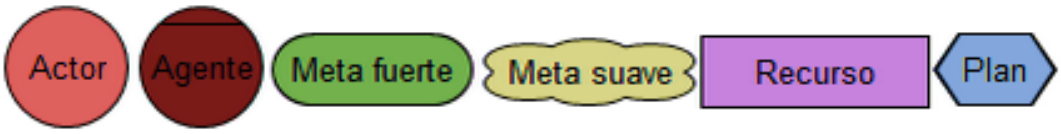



Figura 3: Notación gráfica de los elementos de i* en la herramienta Taom4E.

**Actor:** es una entidad que tiene metas estratégicas e intenciones dentro del sistema o dentro del conjunto organizacional [23].

**Agente (*Agent*):** es un actor con manifestaciones físicas concretas como las de un humano. Es utilizado el término de agente, en lugar de persona, para generalizar su utilización [23].

**Meta fuerte/Meta suave (*hardgoal*)/(*softgoal)*:** estas metas representan los intereses estratégicos de un actor. Las metas fuertes se distinguen de los metas suaves porque en las segundas no existe un criterio claro para definir si han sido satisfechas [23]. Las metas fuertes (*hardgoals)* representan metas a realizar por

un actor, sin embargo las metas suaves (*softgoals)* son metas que representan intenciones que favorecen la realización de una meta.

**Recurso (*Resource*):** representa una entidad física o de información [23].

**Plan:** representa una forma de hacer algo en un nivel abstracto. La ejecución del plan puede ser una manera de satisfacer una meta fuerte o una meta suave [23].

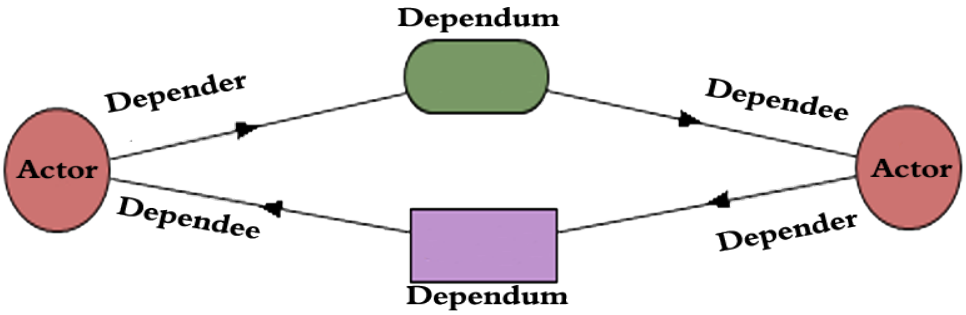



Figura 4: Modelo SD de dependencia entre actores.

**Dependencia:** representa la relación entre dos actores, lo cual indica que un actor depende, por alguna razón, de otro actor para cumplimentar una meta, ejecutar un plan u obtener un recurso. El actor que depende de otro actor es llamado "*depender*", mientras que el actor del cual se depende es llamado "*dependee*". El objeto alrededor del cual se genera esta dependencia es llamado "*dependum*". En la Figura 4 se muestra un ejemplo de dependencia entre actores, en este caso es un modelo SD. La dependencia es intencional si el *dependum* está relacionado de alguna manera a la meta o a un deseo del *depender* [175]. Existen dependencias dentro del actor, en el modelo SR, que se utilizan para desglosar las metas, representar las contribuciones y las dependencias de medio-fin. La Figura 5 muestra un ejemplo de las dependencias dentro de un actor y de estas con otro actor, en este caso usando un modelo SR.

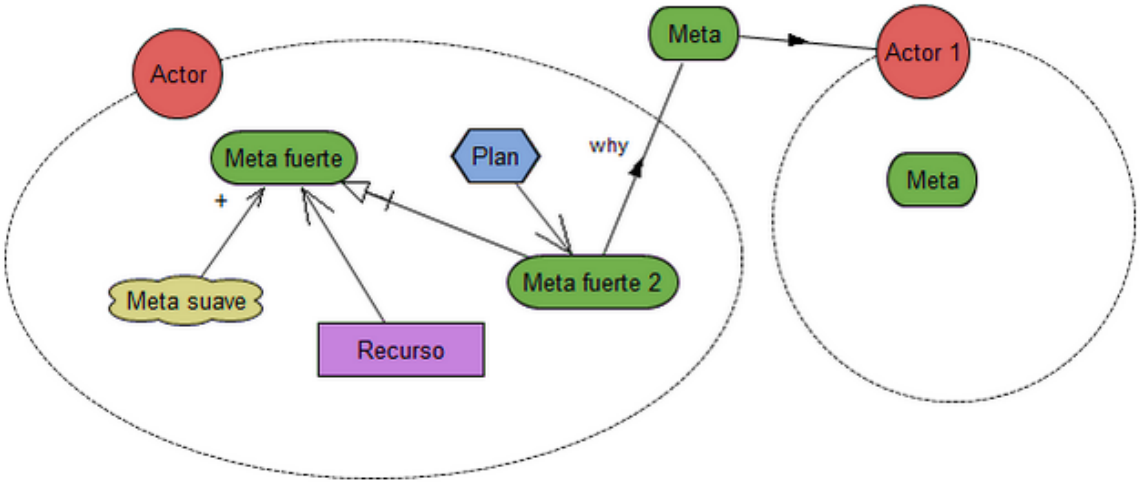



Figura 5: Modelo SR de relaciones internas del actor.

## 1.8 Plataforma para el desarrollo de agentes JADE

Para el desarrollo de sistemas multi-agente existen varias plataformas de desarrollo. JADE[2] está entre las más conocidas y utilizadas debido a las funcionalidades que permite desarrollar, entre las que se encuentran el desarrollo de aplicaciones de agentes que cumplen con las especificaciones FIPA, para sistemas multi-agente y el desarrollo agentes BDI [14, 15, 50, 52].

La plataforma de agentes JADE permite el funcionamiento de un sistema de agentes distribuidos utilizando el lenguaje Java. De acuerdo con el enfoque de los sistemas multi-agente, una aplicación sobre la base de la plataforma JADE se compone de un conjunto de agentes cooperantes que se pueden comunicar entre sí a través del intercambio de mensajes. Cada agente está inmerso en un ambiente sobre el que puede actuar y cuyos acontecimientos puede percibir. El ambiente puede evolucionar de forma dinámica y los agentes aparecen y desaparecen en el sistema de acuerdo a las necesidades y los requisitos de las aplicaciones. JADE proporciona los servicios básicos necesarios para la distribución de aplicaciones en el ambiente, permitiendo a cada agente descubrir a otros de forma dinámica y comunicarse con ellos [15].

[2] Java Agent DEvelopment Framework

JADE es interoperable, ya que al ser compatible con las especificaciones de FIPA [50, 52], sus agentes pueden interactuar con otros agentes siempre y cuando cumplan con el mismo estándar [15].

La plataforma maneja un gran número de elementos por lo que puede resultar complejo su uso. La forma de trabajar con los mensajes y de localizar los agentes se dificulta debido a la cantidad de líneas de código que hay que escribir, lo mismo ocurre para crear los agentes y sus contenedores. Si la herramienta se desea utilizar hay que aprender elementos sobre los agentes, su forma de interactuar y los protocolos FIPA [51] que utiliza. Tanto para los conocedores del tema de agente como para los que no lo son, lleva un gran esfuerzo trabajar con JADE [121]. Es importante destacar que a pesar de lo expuesto anteriormente JADE es una plataforma de desarrollo de agente potente y muy utilizada [158].

## 1.9 Patrones

El desarrollo de software basado en patrones y modelos está rehaciendo el mundo de los desarrolladores de software [149].

De acuerdo con el diccionario de inglés Oxford[3], un patrón “es una forma lógica o regular”, “un modelo”, “el diseño o las instrucciones para hacer algo” o “un ejemplo excelente”. Todos estos significados se aplican en el desarrollo de software, aunque según [44] la última definición es la más acertada.

Christopher Alexander [5] expone que “cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, entonces describen 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”.

Según la clasificación dada por Hevner sobre los artefactos que se obtienen en una investigación, en la Ciencias del Diseño en los sistemas de información [85, 86], un patrón proporciona elementos generalizados de diseño del sistema, que pueden usarse para diferentes tipos de diseños de sistemas. En este sentido, no

[3] Oxford: Oxford English Dictionary http://www.oed.com/

se usan para describir cómo debe ser un sistema específico, un patrón apoya la creación de ese diseño. Los patrones existen para la programación, la arquitectura de sistemas, la arquitectura empresarial, el diseño organizacional, etc. Usualmente, los patrones se describen con sus beneficios y el contexto de aplicación [130].

Los patrones, según la disciplina de la Ingeniería de Software en que se manifiesta el problema que resuelven, pueden ser de diseño, de implementación, etc [13, 63].

“Los patrones de diseño son descripciones de las comunicaciones entre objetos y clases que son personalizables para resolver un problema general de diseño en un contexto particular” [63].

Un patrón de implementación es un módulo de software único en un lenguaje de programación en particular. Una característica crucial es que son fácilmente reconocibles por el software, lo que facilita la automatización [13, 70].

Según Beck “Los patrones de implementación proveen un catálogo de problemas comunes en programación y la forma en que […] se pueden resolver estos problemas” [13].

En general, un patrón tiene cuatro elementos esenciales [63]:

**Nombre:** es un indicador que se usa para describir un problema, sus soluciones y consecuencias.

**El problema:** describe cuándo aplicar el patrón, explicando el problema y su contexto.

**La solución:** se compone de los elementos para solucionar el problema, sus relaciones, responsabilidades y las colaboraciones.

**Las consecuencias:** son los resultados y los cambios resultantes de aplicar el patrón, lo que incluye su impacto sobre la flexibilidad, la extensibilidad o la portabilidad de un sistema.

Los patrones como idea y principio se pueden utilizar tanto en la orientación a objetos [63] como en la orientación a agentes [29, 145, 147]. En la orientación a

objetos los patrones más conocidos y utilizados son los patrones de diseño, ya que los mismos se pueden llevar hasta la implementación.

### 1.10.1 Patrones de diseño en la orientación a objetos

Los patrones de diseño pueden ser de diferentes tipos, dentro de los cuales se encuentran los creacionales, los estructurales y los de comportamiento [63, 149].

En este punto se quiere hacer énfasis en el patrón *Observer*, conocido también por Dependencia o Publicación-Suscripción, que es un patrón de comportamiento. Este patrón define una dependencia de uno a muchos entre objetos, de modo que cuando un objeto cambia de estado, todas sus dependencias son notificadas y actualizadas automáticamente.

La Figura 6 a) muestra un caso típico de empleo del patrón *Observer*. Se trata de un centro de noticias al que están inscritos usuarios con sus preferencias. Al recibir noticias nuevas, el centro distribuye las mismas según la preferencia de los usuarios inscritos.

En la Figura 6 b) se muestra el flujo de trabajo de los componentes en el ciclo de comportamiento del patrón *Observer*. Se inicia con la adición de observadores, luego la instancia *Observable* monitorea el entorno para ver si han ocurrido cambios. Si existe algo que deba ser notificado a los observadores, se envía un mensaje con los datos necesarios para que las instancias de observadores actúen en consecuencia. Este proceso se repite periódicamente en el tiempo. Aunque el principio del patrón *Observer* es mantener informado a un conjunto de elementos de los cambios que desean conocer, esto no se logra de una forma fluida debido al principio de los objetos de sólo responder a peticiones, lo que no permite que el comportamiento sea proactivo. Es importante destacar que el patrón *Observer* está detallado a nivel de diseño y no de implementación, además que la inicialización de la observación no se hace de forma proactiva, es decir, cuando se produce un cambio en el ambiente [149].

En la orientación a objetos también se han desarrollado patrones de requisitos, pero tampoco han propuesto vías para detectar posibilidades de comportamientos

proactivos. Los mismo se enfocan fundamentalmente en la forma en que los requisitos deben escribirse, para que sean más precisos [3, 169].

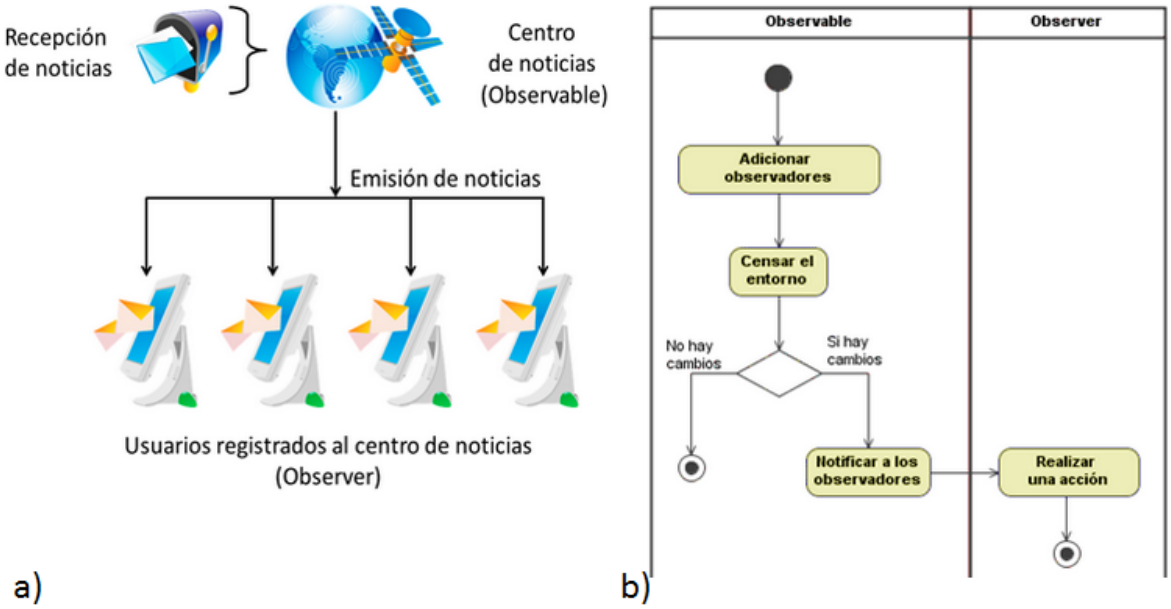



Figura 6: Patrón Observer.

### 1.10.2 Patrones en la orientación a agentes

En la orientación a agentes se han propuesto patrones de diseño para resolver varios problemas propios de los sistemas multi-agente.

Uno de los primeros trabajos es propuesto en [9] y está enfocado en agentes móviles con Aglets[4]. Ese trabajo incluye tres clasificaciones muy orientadas a agentes móviles. Están los patrones de viaje (*traveling*) que están relacionados con el reenvío y enrutamiento de los agentes móviles, patrones de tareas (*task*) para estructurar el trabajo con los agentes, y patrones de interacción (*interaction*) para localizar y facilitar las interacciones entre los agentes. Los patrones están desarrollados en Java y enfocados en el diseño. Utilizan diagramas de clases y de interacción para exponerlos y explicarlos.

En ese trabajo se presentan dos aplicaciones basadas en agentes (*File Searcher* y *Enhanced File Searcher*), donde se emplea la combinación de patrones. Esas

[4]Aglets http://www.research.ibm.com/trl/aglets/

aplicaciones se utilizan para la implementación de agentes móviles que buscan archivos con cierta cadena en el nombre y que pueden viajar por varios servidores para hacer la búsqueda. En ambos casos se basan en una filosofía reactiva donde los agentes buscan lo que le pide un "*master*" y lo devuelven, pero no conservan ninguna memoria de esa búsqueda. Tampoco se modela una solución a la gestión de cambios en los ficheros almacenados en los servidores sin necesidad de volver a enviar la búsqueda [9].

En [147] y [148] se enfatiza en la necesidad de los patrones de diseño en la orientación a agentes, como forma de recolectar y formalizar experiencias para soluciones basadas en este paradigma. En ese trabajo se definen cuatro clases de patrones: metapatrones, patrones metafóricos, patrones arquitecturales y antipatrones. Siguiendo esta clasificación se desarrolla una propuesta de once patrones. Según Sauvage [146, 147] muchos patrones orientados a agentes son realmente patrones orientados a objetos, ya que no van a aspectos singulares de la orientación a agentes, como la autonomía y las interacciones. Además expresa que muchos patrones en la orientación a agentes se enfocan en el diseño, obviando la importancia de tener patrones orientados a agentes en otras dimensiones como el análisis o la implementación.

En [131] se presenta un esquema de clasificación bidimensional de los patrones. En esa clasificación según el aspecto de diseño (clasificación horizontal), están los estructurales, de comportamiento, sociales y de movilidad. Según el nivel de diseño (clasificación vertical), están los patrones de análisis de roles, patrones de diseño de agentes, patrones de diseño de sistema, patrones de la arquitectura del agente y los patrones de implementación del agente.

Un mérito importante de ese trabajo es que su clasificación es amplia y cubre varios niveles de abstracción. Aunque los patrones se presentan en términos de los conceptos de la metodología ROADMAP [16], lo hace de una forma abarcadora y general. Se exponen algunos ejemplos de patrones de agentes y sus clasificaciones. Se enfatiza en que esta clasificación se enfoca más en las nociones del paradigma de agentes. Entre los campos que sugieren para describir

los patrones se encuentran: el aspecto de diseño (clasificación horizontal) y el nivel de diseño (clasificación vertical) [131]. La Tabla 2 muestra un esquema de las dos clasificaciones de los patrones en los agentes y las X representan los patrones de agentes que existen según esa clasificación [131].

Entre los patrones que propone [131] está el *Ecological Recogniser,* el cual trata de inferir las intenciones de los agentes y se enfoca en el descubrimiento.

Tabla 2: Clasificación de los patrones en la orientación a agentes.

| **Horizontal** / **Vertical** | **Estructural** | **Comportamiento** | **Social** | **Movilidad** |
|---|---|---|---|---|
| Análisis de roles | | | X | X |
| Diseño de agentes | X | X | X | X |
| Diseño del sistema | X | | X | X |
| Diseño del agente interno | X | X | | X |
| Implementación del agente | X | X | X | X |

Existen otros trabajos, tal es el caso del repositorio de patrones propuesto por el grupo de desarrollo de la metodología PASSI que propone un conjunto de patrones, entre los que se encuentran [29, 33]:

-patrones multi-agente que están relacionados con la colaboración entre dos o más agentes;

-patrones para un solo agente, donde se propone una solución para la estructura interna de un agente junto con sus planes de realización de un servicio específico;

-patrones de comportamiento que proponen una solución para agregar una capacidad específica al agente;

-patrones de especificación de acciones que agregan una funcionalidad simple al agente.

Todos estos patrones son para desarrollar un sistema multi-agente más robusto.

Sabatucci en el trabajo [145] se enfoca en patrones de diseño y defiende que un aspecto importante es no usar un solo patrón sino combinarlos. En ese trabajo se hace la formalización de los patrones con un lenguaje que favorece la combinación y están integrados a PASSI [82].

En ninguno de los patrones que se describen en los trabajos anteriormente mencionados se hace énfasis en la proactividad o en ambientes a observar, sino en otras propiedades como la cooperación, la comunicación, la estructura organizacional de los agentes u otras. En la orientación a agentes no se han desarrollado patrones de requisitos que brinden vías para detectar posibilidades de comportamientos proactivos. La mayoría de estos patrones se enfocan en el diseño y no en la implementación. En esta dirección, tampoco se conoce de ningún trabajo enfocado en simplificar el trabajo con JADE [14], encapsulando la solución de problemas comunes en la construcción de un SMA.

Particularmente estos dos aspectos (la proactividad y la simplificación de la configuración de JADE) son dos problemas comunes en muchas soluciones basadas en SMA, para las cuales no se conocen que hayan patrones de implementación definidos.

## 1.10 Modelo-V para las pruebas

En el desarrollo de un software siempre hay que tener en cuenta la calidad con que se construye el mismo y la calidad final del mismo [163]. Las pruebas forman parte del proceso de asegurar la calidad de un producto de software.

Una definición clásica de prueba de software se puede encontrar en [124] donde se plantea que “la prueba es el proceso de ejecutar un programa con la intención de encontrar los errores”. Durante los últimos años, la forma de ver las pruebas ha evolucionado. La prueba de software es vista ahora como un proceso completo que apoya el desarrollo y actividades de mantenimiento. Las pruebas pueden derivarse de los requisitos y especificaciones, de los artefactos de diseño o del código.

Un nivel diferente de prueba acompaña cada actividad del desarrollo de software [7]. Dependiendo de las actividades del ciclo de vida del software, pueden definirse diferentes tipos de pruebas. Esto se ve en Modelo-V [1] que se muestra en la Figura 7. La rama izquierda de la V representa el flujo de trabajo y la rama derecha, representa el flujo de prueba, donde el producto de software es probado en los diferentes niveles de abstracción [7].

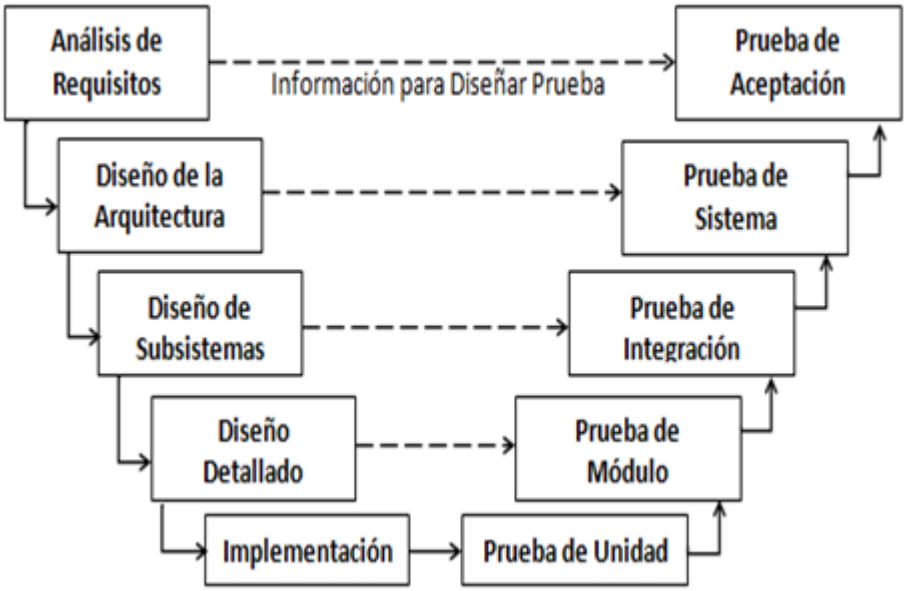



Figura 7: Actividades en el desarrollo de software y los niveles de prueba en el “Modelo-V”.

La información de cada nivel de prueba se deriva normalmente de la actividad de desarrollo relacionada. Un consejo importante que se da es que se diseñen las pruebas simultáneamente con cada actividad de desarrollo, aunque el software no esté aún listo para ejecutarse [90].

El propósito de la prueba de aceptación es determinar si el software final satisface los requisitos del sistema. La prueba de sistema intenta determinar si el sistema cumple sus especificaciones. La prueba de integración es para evaluar la comunicación entre los módulos en un subsistema [122]. La unidad de un programa es una o más declaraciones del programa, con un nombre que otras partes del software utiliza [124, 140, 156]. Un módulo es una colección de unidades relacionadas que se agrupan en un archivo. El propósito de la prueba de

módulo es evaluar los módulos individuales en el aislamiento, observando cómo las unidades interactúan con otras y las estructuras de datos asociadas. Al nivel más bajo, la prueba de unidad es para evaluar las unidades desarrolladas en la fase de implementación.

Como se puede ver, la forma de concebir las pruebas con el Modelo-V permite utilizarlo en software con diferentes paradigmas de programación. En la orientación a objetos se utiliza el Modelo-V [7]. En la orientación a agentes, a pesar de que la fase de prueba no se ha trabajado con profundidad, debido a las características intrínsecas de los agentes, el Modelo-V se ha utilizado en trabajos para proponer actividades de pruebas en los SMA [122, 126, 127, 128].

Utilizar la simulación del sistema multi-agente para medir los estados del sistema con el fin de examinar las cualidades observables y explorar las propiedades esperadas del sistema en su conjunto se ha propuesto por algunos trabajos de pruebas de agentes [126, 127, 161]. La simulación es un proceso que puede ser muy costoso, consumir mucho tiempo inicialmente y construir los modelos de simulación precisa de un entrenamiento especial [11, 100]. Para evaluar y ayudar el diseño de un sistema multi-agente algunos trabajos proponen utilizar simulación basada en agentes [27, 71].

## 1.11 Conclusiones Parciales

Las principales conclusiones que se pueden extraer de este capítulo a partir de la revisión bibliográfica realizada son:

- La proactividad permite tener software dirigidos a metas y de esta forma tomar iniciativas.
- Las metodologías orientadas a objetos y las orientadas a agentes no presentan pasos para extraer requisitos proactivos para un software. Particularmente, no se conoce ningún patrón de requisitos que permita encontrar posibilidades de proactividad.

- El lenguaje de modelado i* está orientado a metas y sigue los principios del modelado social, lo que le permite modelar las dependencias entre los actores estratégicos y sus intenciones.
- Para detectar la proactividad se necesitan conocer las causa de las dependencias (los por qué), y la intencionalidad de las relaciones. Esta información está contenida en los modelos de i*.
- JADE es una plataforma de agentes robusta y con muchas facilidades para el desarrollo de un SMA. En esta dirección, no se conoce de ningún trabajo enfocado en simplificar el trabajo con JADE, encapsulando la solución de problemas comunes en la construcción de un SMA.
- Los patrones en la orientación a agentes no hacen énfasis en la proactividad. La mayoría de estos patrones se enfocan en el diseño y no en la implementación.
- El Modelo-V se puede utilizar para plasmar las actividades de pruebas y el ciclo de vida de desarrollo de un software sin importar el paradigma.

# *Capítulo 2*

# *Patrones de requisitos e implementación para incorporar proactividad*

# *Capítulo 2: Patrones de requisitos e implementación para incorporar proactividad*

## 2.1 Introducción

En este capítulo se introducen cuatro patrones, primero se describen dos patrones de requisitos para detectar proactividad en los sistemas informáticos a partir de los requisitos tempranos plasmados en i* y delegarlos en el software a desarrollar reflejado en los requisitos tardíos de i*. También se proponen dos patrones de implementación para construir un sistema que desee manejar agentes JADE y tener comportamientos proactivos a partir de la observación periódica de un ambiente. Además se describen actividades de pruebas que se derivan de la fase de captura requisitos en i* y de desarrollar agentes con los patrones de implementación.

## 2.2 Comportamiento proactivos a partir de los requisitos en i*

Como se expuso en el capítulo 1, el lenguaje de modelado i* adopta una visión social del mundo donde se puede ver 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. En el mundo, los actores no existen en forma aislada, están situados en algún entorno compartiendo e interactuando con otros actores.

Para detectar la proactividad se necesita conocer las causa de las dependencia (los por qué), la intencionalidad de las relaciones. A partir del modelado de los requisitos en i* es posible detectar metas a cumplimentarse proactivamente. Estas intenciones se detectan en los requisitos tempranos. Según Yu [175], la dependencia es intencional si el *depempum* (objeto del que se depende) está relacionado de alguna manera a la meta o el deseo del *depender* (actor que depende). Las intenciones se pueden delegar al software y lograr de esta forma que él mismo trabaje en pos de lograr metas e intenciones, los que conlleva a un comportamiento proactivo.

## 2.3 Patrones de requisitos para detectar proactividad

A continuación se presentan dos patrones de requisitos que utilizan como base los modelos de i* en los requisitos tempranos y tardíos. Ambos tratan un problema en común: actores en los requisitos tempranos con intenciones que denotan la necesidad de un comportamiento proactivo. Ambos describen la solución para delegar estas intenciones, de los actores en los requisitos tempranos hacia el software que se desarrollará, reflejado en los requisitos tardíos. Es importante aclarar que ambos patrones permiten detectar posibilidades de proactividad, lo cual no quiere decir que siempre sea necesario, fácil o conveniente que los sistemas a implementar incluyan estos requisitos. Es decir, estos patrones de requisitos ayudan a detectar un posible comportamiento proactivo, pero como cualquier requisito debe ser aceptado por los implicados en el desarrollo del sistema.

Los patrones de requisitos que se describen tienen en común que exista una dependencia entre dos actores: uno que representa una entidad de software o proceso a automatizar y otro actor que se beneficiaría con un comportamiento proactivo del software. Este segundo actor beneficiario de la proactividad puede representar a un humano o también una entidad de software y se le nombrará “Beneficiario” y al primero se le llamará “Software”.

En ambos patrones, en la descripción del problema, el actor “Beneficiario” depende del actor “Software” a través de una meta fuerte para cumplimentar una meta propia. El actor “Beneficiario” tiene una meta suave, que representa una intención que contribuye positivamente a la meta propia del mismo. La diferencia esencial entre ambos es que en uno la dependencia está asociada a una meta, mientras que en el otro está asociada a un recurso

En los dos patrones el problema se presenta en los requisitos tempranos del modelo i*. Mientras que la solución se plasma en un modelo de requisitos tardíos también de i*. Es importante aclarar que tanto el problema como la solución pueden constituir partes de un modelo de i* mayor. Es decir, lo que se describe es

parte de una situación posiblemente más grande, de la cual solo se está haciendo énfasis en un subconjunto de interés.

Se ha preferido usar nombres en inglés para describir los patrones debido a ser este lenguaje el más comúnmente usado para describir patrones [13, 63, 145].

### 2.3.1 Patrón Hardgoal why Dependency

El patrón de requisitos *Hardgoal why Dependency* se puede utilizar cuando se desea desarrollar un software con comportamientos proactivos y existen dependencias de metas entre actores estratégicos del proceso que se quiere automatizar. A continuación se detallan los elementos esenciales del patrón:

**Nombre del patrón:** *Hardgoal 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 presentan el estado que se describe a continuación, este estado puede representar un subconjunto del modelo de requisitos tempranos:

- Existe un actor “Beneficiario” que tiene una meta fuerte “meta principal” (siguiendo ejemplo de la Figura 8). A dicha meta fuerte le contribuye positivamente una meta suave “meta suave/intención”. Ambas metas son sólo del actor “Beneficiario”, no se derivan de una dependencia.
- Existe una dependencia entre el actor “Beneficiario” y el actor “Software” de una meta fuerte “meta y” para cumplimentar la meta fuerte “meta principal” del actor “Beneficiario”.
- El actor “Software” para cumplimentar la meta fuerte “meta y” depende del actor “Beneficiario” por una “meta x”.
- La “meta x” contribuye positivamente a la intención o meta suave “meta suave/intención” del actor “Beneficiario”.

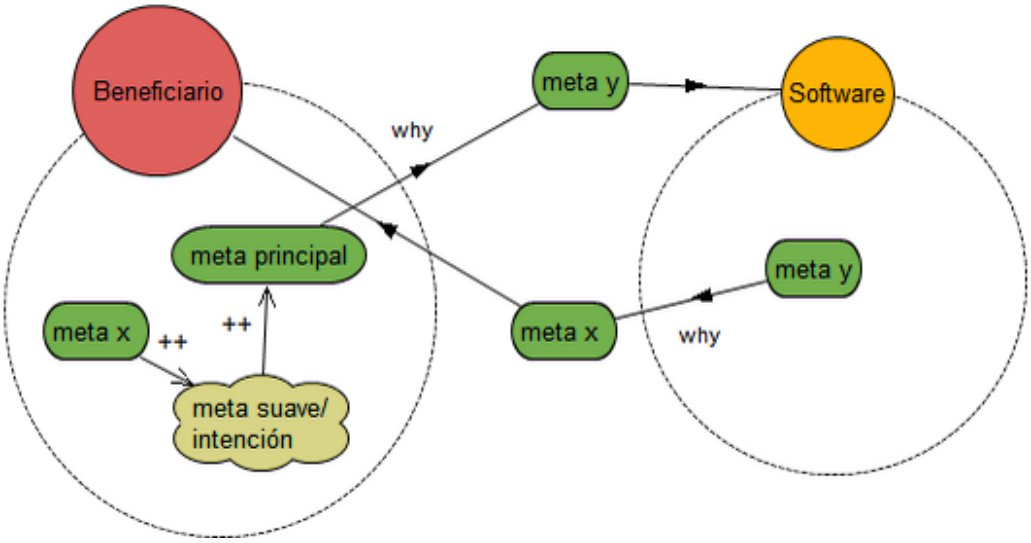



Figura 8: Subconjunto del modelo de requisitos tempranos con i*que representa el problema del patrón *Hardgoal why Dependency*.

**Solución:** Para delegar las intenciones en el software que se pretende desarrollar se construye un modelo o subconjunto del modelo de requisitos tardíos en i* donde se debe:

- Transformar al actor "Software" en el sistema a desarrollar con comportamiento proactivo, por lo que se presenta como un agente.
- Delegar al actor "Software" la meta suave "meta suave/intención", que representa la intención.
- Delegar al actor "Software" la meta fuerte "meta x".
- Transformar la meta fuerte "meta x" en un plan "meta x", que es un medio para cumplimentar la meta suave "meta suave/intención".
- Reflejar que la meta suave "meta suave/intención" contribuye positivamente a la meta fuerte "meta y" que se realiza para cumplimentar la meta fuerte "meta principal" del actor "Beneficiario".

La Figura 9 muestra la solución del patrón *Hardgoal why Dependency* plasmada en un subconjunto del modelo 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 *Hardgoal why Dependency* se reflejarán en los requisitos tardíos siguiendo los pasos de i*.

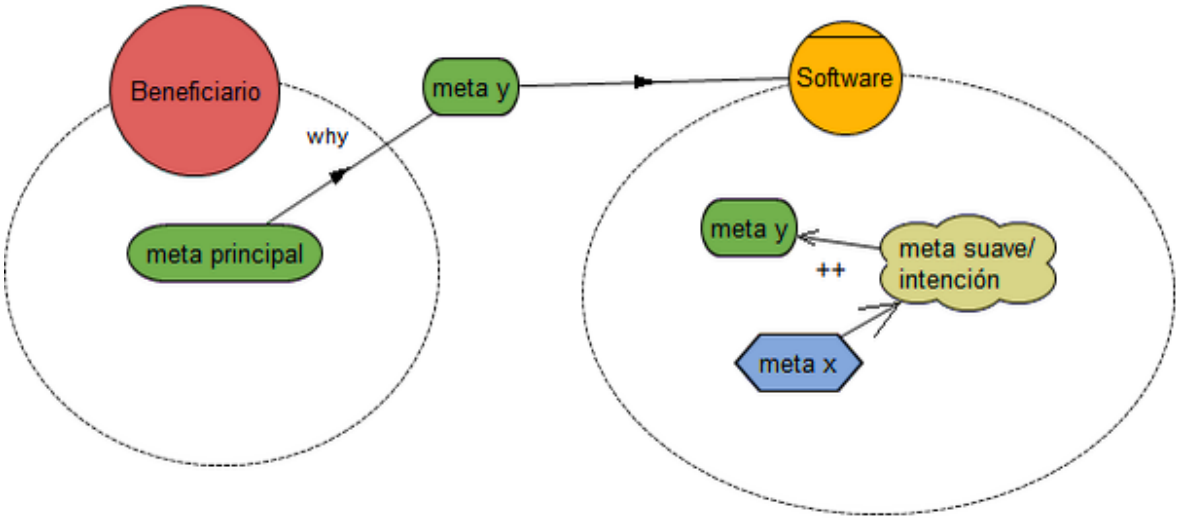



Figura 9: Subconjunto del modelo de requisitos tardíos con i* que representa la solución del patrón *Hardgoal why Dependency*.

**Consecuencia:** Este patrón permite delegar intenciones al software para que este cumplimente sus metas proactivamente. Su objetivo es dar una vía para delegar dichas intenciones cuando se desea tener proactividad en un software y pueda reportar beneficios en el uso del mismo. La meta se ha convertido en un plan, el mismo necesita ser implementado para que pueda haber proactividad.

### 2.3.2 Patrón Resource why Dependency

El patrón de requisito *Resource why Dependency* se puede utilizar cuando se desea desarrollar un software con comportamientos proactivos y existen dependencias de recursos entre actores estratégicos del proceso que se quiere automatizar. A continuación se detallan los elementos esenciales del patrón:

**Nombre del patrón:** *Resource 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 presentan el siguiente estado que se describe, este estado puede representar un subconjunto del modelo de requisitos tempranos:

- Existe un actor “Beneficiario” que tiene una meta fuerte “meta principal” (siguiendo ejemplo de la Figura 10). A dicha meta fuerte le contribuye positivamente una meta suave “meta suave/intención”. Ambas metas son sólo del actor “Beneficiario”, no se derivan de una dependencia.
- Existe una dependencia entre el actor “Beneficiario” y el actor “Software” de una meta fuerte “meta z” para cumplimentar la meta “meta principal” del actor “Beneficiario”.
- El actor “Software” para cumplimentar la meta fuerte “meta z” depende del actor “Beneficiario” por un “recurso x”.
- El “recurso x” es un medio para cumplimentar la intención o meta suave “meta suave/intención”.

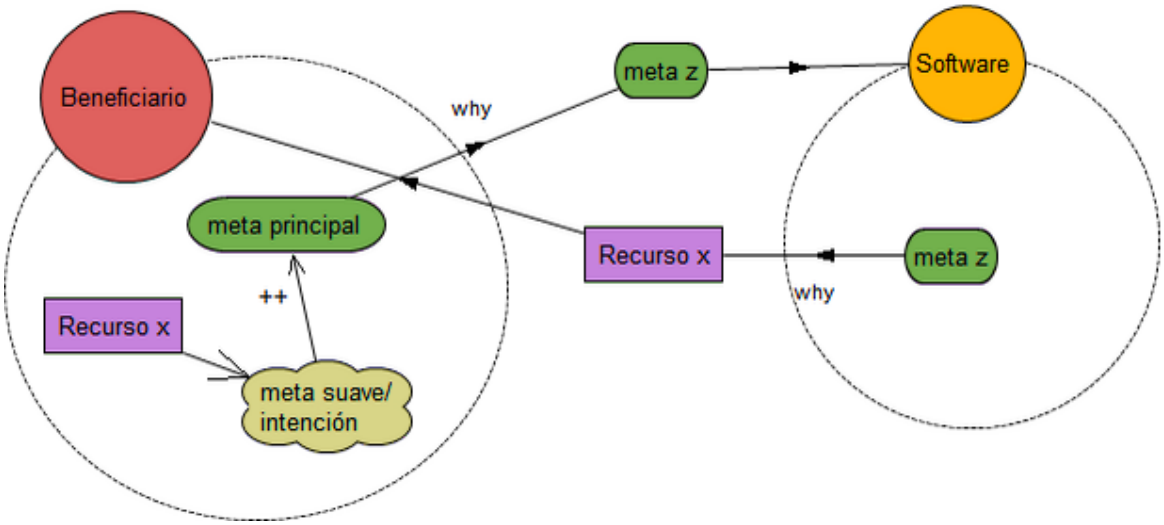



Figura 10: Subconjunto del modelo de requisitos tempranos con i* que representa el problema del patrón *Resource why Dependency*.

**Solución:** Para delegar las intenciones en el software que se pretende desarrollar se construye un modelo o subconjunto del modelo de requisitos tardíos en i* donde se debe:

- Transformar al actor “Software” en el sistema a desarrollar con comportamiento proactivo, se presenta como un agente.

- Delegar al actor “Software” la meta suave “meta suave/intención”, que representa una intención.
- Delegar al actor “Software” el recurso “recurso x”. En este caso el recurso “recurso x” seguirá siendo un medio para cumplimentar la meta suave “meta suave/intención”, pero además pasará a ser un medio para cumplimentar la meta fuerte “meta z”.
- Reflejar que la meta suave “meta suave/intención” contribuye positivamente a la meta fuerte “meta z” que se realiza para cumplimentar la meta fuerte “meta principal” del actor “Beneficiario”.

La Figura 11 muestra la solución del patrón *Resource why Dependency* plasmada en un subconjunto del modelo 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 *Resource why Dependency* se reflejarán en los requisitos tardíos siguiendo los pasos de i*.

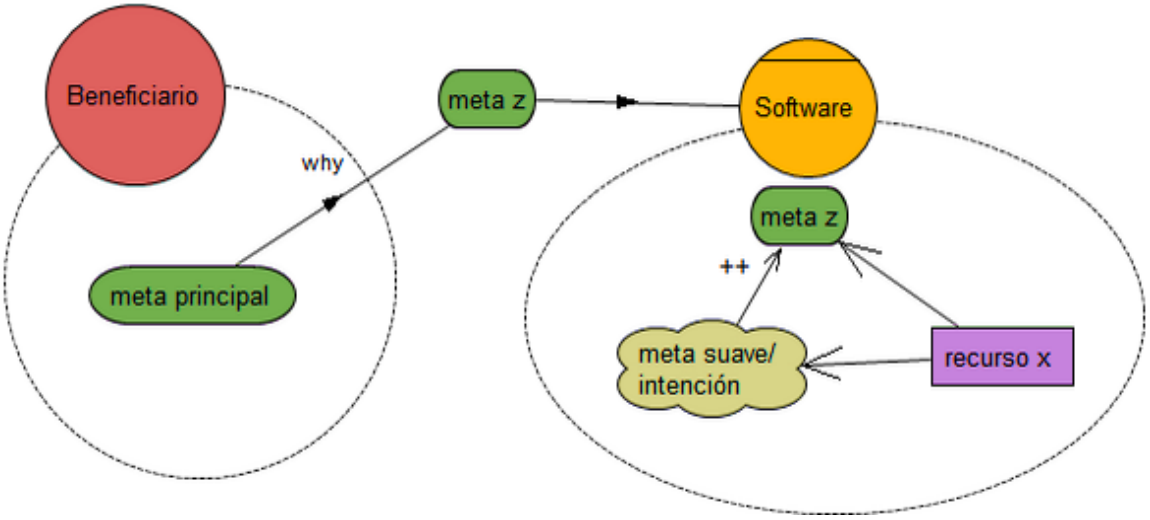



Figura 11: Subconjunto del modelo de requisitos tardíos con i* que representa la solución del patrón *Resource why Dependency*.

**Consecuencia:** Este patrón permite delegar intenciones al software para que este cumplimente sus metas proactivamente. Su objetivo es dar una vía para delegar dichas intenciones cuando se desea tener proactividad en un software y reportar

beneficios en el uso del mismo. En ocasiones es necesario implementar un plan que se encargue de manejar el recurso.

### 2.3.3 Patrón Hardgoal why Dependency y el patrón Resource why Dependency

Como se puede apreciar la diferencia en el problema entre el patrón *Hardgoal why Dependency* y el patrón *Resource why Dependency* radica en que: en el patrón *Hardgoal why Dependency* existe una dependencia de meta fuerte y en el patrón *Resource why Dependency* existe una dependencia de recurso. La solución en los dos patrones de requisitos es delegar en el actor “Software”, que se convierte en un agente, una meta o un recurso para tener un comportamiento proactivo. Puede existir un modelo de requisitos tempranos en i* en que se encuentren ambos problemas tratados por ambos patrones.

En el caso de que exista un modelo que presente el problema del patrón *Hardgoal why Dependency* y el problema del patrón *Resource why Dependency* se puede utilizar la solución de ambos patrones y delegar las metas y los recursos al software a desarrollar. La utilización de ambos patrones así como la combinación de los dos se presenta en uno de los casos discutidos en el capítulo 3.

Al terminar la aplicación de los patrones o patrón se puede seguir el desarrollo del sistema es su fase de diseño con la metodología Tropos, ya que el modelo de requisitos tardíos de i* es el modelo (artefacto) de entrada en Tropos para su fase Diseño de la Arquitectura [115]. En caso de que se quiera seguir el desarrollo con otra metodología de agentes como Ingenias existe una herramienta que permite la transformación del modelo de requisitos tardíos de i* hacia la fase de análisis de Ingenias para incorporar el enfoque del modelado social en esta robusta metodología y desarrollar un software orientado a agentes [81].

También existen los pasos para incorporar la visión del modelado social en la orientación a objetos. Para esto se desarrolló una herramienta que transforma los modelos obtenidos en i* hacia *UML* en el Modelo del Negocio y el Modelo de Requisitos en *RUP*. Esta herramienta permite seguir el desarrollo de un software orientado a objetos con *RUP* a partir de una captura de requisitos en i* [81, 112].

## 2.4 Patrones de implementación para incorporar proactividad

En esta sección se proponen dos patrones de implementación: el patrón *Implementation_JADE* que se enfoca en simplificar la configuración de JADE para crear y manejar agentes; y el patrón *Proactive Observer_JADE* que se enfoca en la incorporación de proactividad. Dichos patrones siguen las recomendaciones de [147] y [131] de que los patrones en la orientación a agentes se enfoquen a las propiedades singulares de los agentes expuestas en el capítulo 1.

Teniendo en cuenta la consolidación alcanzada por JADE como plataforma de software libre para el despliegue de un sistema multi-agente y que está basada en el estándar FIPA [54], se decidió utilizar dicha plataforma para la propuesta de los patrones. Un patrón de implementación debe estar hecho en un lenguaje de programación específico [13], se utiliza el lenguaje Java que es empleado por JADE.

El patrón *Implementation_JADE* se enfoca en los principios expuesto por Beck [13], en cuanto a que los patrones de implementación permitan de que los programadores se enfoquen en lo que es realmente singular de cada problema, ya que encapsula parte de la complejidad del trabajo con JADE.

El patrón *Proactive Observer_JADE* sigue la sugerencia de Sabatucci y otros [145] de enfatizar en la composición de patrones para crear nuevos patrones. Este patrón guarda relación con el patrón *Ecological Recogniser* abordado en [131]. En el caso del *Proactive Observer_JADE* las intenciones se conocen y se relacionan con un ambiente que se observa.

En la presentación de ambos patrones se incluyen los campos clasificación horizontal y clasificación vertical sugeridos en [131].

### 2.4.1 Patrón *Implementation_JADE*

El patrón *Implementation_JADE* simplifica el uso y la configuración de los aspectos principales para el trabajo en la plataforma JADE. A continuación se detallan los elementos esenciales del patrón:

**Nombre del patrón:** *Implementation_JADE*.

**Problema**: El patrón que aquí se describe se debe utilizar cuando se quiera implementar las características más comunes y que son de vital importancia para la ejecución del sistema multi-agente. Este patrón se puede utilizar cuando se desea hibridar un software tradicional (orientado a objetos) con uno o varios agentes. También se puede utilizar cuando se quiere construir un sistema multi-agente que necesite tener cambios en el comportamiento de los agentes y crear o eliminar agentes en tiempo de ejecución. Entre las características más comunes para la ejecución del sistema multi-agente están las siguientes:

1. Ejecutar y controlar la plataforma que contiene a los agentes.
2. Gestionar el ciclo de vida de los agentes.
3. Comunicar los agentes que viven dentro del sistema.
    3.1. Enviar y recibir mensajes desde y hacia otros agentes.
    3.2. Procesar el mensaje y tomar acciones dependiendo de su contenido.
4. Comunicar con los agentes coordinadores de la plataforma.
    4.1. Ver el estado de un módulo del sistema.
    4.2. Buscar un agente para ver su estado.

**Solución**: Este patrón utiliza la plataforma JADE para el trabajo con los agentes, sirviendo como intermediario a las funcionalidades que implementa JADE. Es importante resaltar que JADE es código de Java, por lo que facilita la creación sistemas híbridos entre objetos y agentes.

El patrón *Implementation_JADE* contiene 9 grupos de operaciones:

1. Inicializar la plataforma de agentes JADE.
    1.1. Configurar parámetros de funcionamiento de la plataforma.
    1.2. Crear los contenedores necesarios para colocar los agentes.
    1.3. Ejecutar los agentes en los contenedores correspondientes.
2. Unir a una plataforma ya existente.
3. Conectar un agente que ha perdido la plataforma que lo maneja.

4. Implementar un comportamiento cíclico para poder procesar los mensajes que recibe un agente determinado.
5. Enviar mensajes hacia los agentes.
    5.1. Configurar parámetros de los mensajes.
6. Trabajar con el agente AMS (*Agent Management Service*).
    6.1. Controlar el funcionamiento de la plataforma.
    6.2. Buscar agentes para comunicarse.
    6.3. Conocer estado de un contenedor.
7. Trabajar con el agente DF (*Directory Facilitator*).
    7.1. Encontrar un agente que cumpla con un atributo determinado.
8. Trabajar con los comportamientos de un Agente.
    8.1. Insertar comportamientos.
    8.2. Eliminar comportamientos.
    8.3. Realizar la búsqueda de un comportamiento mediante su nombre.
9. Trabajar con los agentes creados.
    9.1. Modificar un agente.
    9.2. Eliminar un agente.

La operación 2 “Unir a una plataforma ya existente” se plantea ya que en ocasiones es necesario tener a los agentes en localidades físicas diferentes, por lo que hay que ejecutar contenedores y agentes en una plataforma que ya existía con anterioridad.

La operación 3 “Conectar un agente que ha perdido la plataforma que lo maneja” se plantea en el escenario en que un agente esté de forma física en una localidad diferente. Puede ser posible que la plataforma colapse y que sea necesario que los agentes sigan trabajando de forma independiente. Luego cuando la plataforma vuelva a funcionar, los agentes pueden reincorporarse a su plataforma

correspondiente y socializar los resultados que obtuvieron mientras trabajaban solos.

La operación 4 “Implementar un comportamiento cíclico…” es necesaria para que el desarrollador pueda decidir qué tipos de mensaje son aceptados. Permite que el desarrollador procese el mensaje como lo desee.

Para implementar las operaciones anteriormente mencionadas se definió un grupo de clases con funcionalidades generales que interactúan y utilizan clases de la plataforma JADE. El diagrama de clases donde se muestran las clases del patrón *Implementation_JADE* se expone en la Figura 12. Las clases “jade” son propias de la plataforma JADE.

*Init_Platform* inicializa la plataforma JADE con las configuraciones que esta permite, como por ejemplo, el puerto de conexión. Se encarga de crear los contenedores en los que se almacenarán los agentes e inicializa los agentes.

*Join_Platform* crea los contenedores y agentes externos, que son subscritos a una plataforma que ha sido inicializada.

*Work_DF* contiene todo el trabajo que se realiza en coordinación con el DF (*Directory Facilitator*), que es similar a las páginas amarillas de una guía telefónica. Registra en el directorio del DF el servicio que un agente tiene, para ofrecer la posibilidad de buscar agentes en esos registros.

*Work_ACL* engloba el trabajo que se realiza con el uso de los mensajes *ACL* (*Agent Communication Language*). Configura los mensajes con los parámetros que son introducidos como: tipo de mensaje, el contenido del mensaje, la identificación de los agentes involucrados en la comunicación, el objeto que se quiere enviar, etc. Cada mensaje enviado contiene un identificador único.

*Work_AMS* contiene todo el trabajo que se realiza en coordinación con el AMS (Agent Management System). Devuelve la descripción del agente que cumple con la condición que el usuario desee.

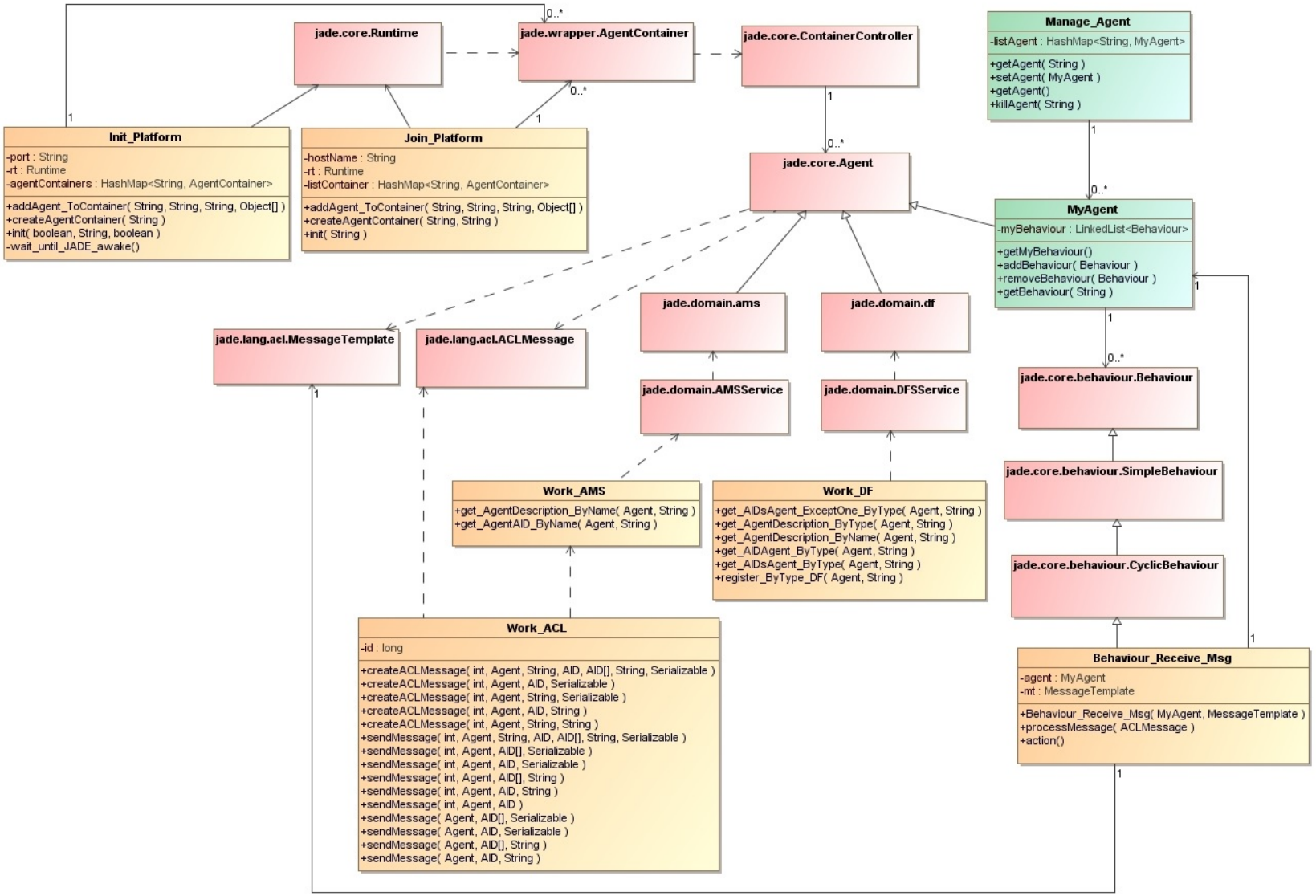



Figura 12: Diagrama de clases del patrón Implementation_JADE.

*Behaviour_Receive_Msg* implementa un *Cyclic Behaviour*[5], para obtener y procesar los mensajes ACL que le llegan al agente al que pertenece. Brinda la posibilidad de extender el método "*processMessage*", que se ejecuta cuando se obtiene un mensaje.

*MyAgent* extiende de Agent, brinda la posibilidad de obtener los comportamientos de un agente así como eliminar, modificar e insertar nuevos comportamientos.

*Manage_Agent* con tiene todos los agentes creados, permitiendo modificar y eliminar los agentes.

**Consecuencias**: Su objetivo es simplificar el desarrollo de un conjunto de agentes, al tiempo que garantiza el cumplimiento de los estándares a través de un amplio conjunto de servicios del sistema. Permite que los desarrolladores puedan hacer uso de la tecnología de agentes para incorporar proactividad a un sistema que no sea basado en agente, haciendo viable un sistema híbrido "agentes+objetos" [84]. Este patrón encapsula la capa de abstracción que facilita la configuración del patrón *Proactive Observer_JADE*.

**Clasificación horizontal**: Estructural y social.

**Clasificación vertical**: Implementación del agente.

### 2.4.2 Patrón *Proactive Observer_JADE*

Este patrón utiliza los principios del patrón de comportamiento *Observer* [63], utiliza como base el protocolo de interacción *Subscribe* de FIPA [54] y los combina con el patrón *Implementation_JADE*.

**Nombre del patrón:** *Proactive Observer_JADE*.

**Problema**: Se utiliza el patrón en cualquiera de las siguientes situaciones :

- Cuando existen dos entidades, una en función de la otra para realizar una acción.

[5]Comportamiento implementado en JADE para efectuar una acción indefinidamente.

- Cuando hay un cambio en una entidad, y es necesario cambiar a las demás pero no se sabe cuántas entidades más habría que cambiar.
- Cuando un entidad debe ser capaz de notificar a otras entidades sin hacer suposiciones acerca de quiénes son estas.
- Cuando una entidad, para cumplimentar una meta, necesita de otra para que encuentre cambios relevantes según sus intereses sin que medie una petición.

**Solución:** Para implementar el patrón se necesitan de dos entidades: "*Observable*" y "*Observer*".

Entre las funcionalidades más relevantes de la entidad "*Observable*" están las siguientes:

1. Percibir el ambiente cada determinado tiempo para detectar algún cambio.
2. Gestionar una lista con los observadores que se subscriban.
3. Notificar a los observadores con los datos encontrados en el cambio ocurrido.

La entidad "*Observer*" tiene las funcionalidades siguientes:

1. Implementar el proceso de subscripción a la lista del "*Observable*".
2. Actualizar su estado interno.
3. Realizar una acción cuando se le notifique un cambio.

Para lograr una implementación genérica del patrón con agentes, se deben crear fundamentalmente dos agentes: *Observer* y *Observable*. Los agentes que cumplirán con estos roles tendrán la capacidad de comunicarse y actuar autónomamente, pudiendo hasta cambiar su comportamiento en dependencia de las situaciones a las que se enfrenten.

El agente con el rol de *Observable* debe tener una lista interna de los agentes *Observer* para poder alertarlos de los cambios que detecta. Además debe esperar los mensajes de subscripción de los agentes *Observer* para poder sumarlos a la lista mencionada y enviar la respuesta de la subscripción. Por último debe tener la

capacidad de enviar un mensaje a los *Observers* con los datos necesarios del cambio encontrado.

El agente *Observer* tiene que conocer los *Observables* existentes en el entorno para decidir a cuáles subscribirse. Además debe actualizar su estado cuando le llegue un mensaje con los datos que describen el cambio ocurrido y actuar en consecuencia.

De forma general se necesita que estos agentes se ejecuten en una infraestructura que gestione los mensajes y el ciclo de vida de los agentes.

Como se describió anteriormente en el patrón *Implementation_JADE* se implementaron un grupo de clases para facilitar el trabajo con la plataforma de agentes JADE. Sobre la base de estas clases se realizaron otras que ejecutan las funcionalidades del patrón *Proactive Observer_JADE* para incluir proactividad. De esta forma, los siguientes pasos deben ejecutarse en el momento de comenzar a utilizar el patrón *Proactive Observer_JADE* creado:

1. Inicializar la plataforma de agentes JADE con la clase *Init_Platform*.
   1.1. Crear los contenedores que contienen a los agentes.
   1.2. Añadir los agentes necesarios a los contenedores
2. Implementar las acciones del *Observable* y los *Observers*. En este caso la clase creada para este fin es *Agent_Actions*.
3. Guardar trazas de la comunicación entre los agentes (opcional).

Se implementó el patrón *Proactive Observer_JADE* utilizando agentes, con las clases que proporciona JADE, logrando que el patrón funcione en un ambiente distribuido.

El diagrama de clases donde se exponen las principales clases del patrón *Proactive Observer_JADE* y su relación con el patrón *Implementation_JADE* se muestra en la Figura 13. A continuación se explican las clases que le dan las funcionalidades necesarias al patrón:

*Agent_Actions* tiene un método *ObserverAction* que se ejecuta cuando al agente *Observer* le llega un mensaje y un método *ObservableAction* que censa el ambiente cada determinado tiempo. La clase está diseñada para que el usuario coloque el código de las acciones que desea realizar en cada caso.

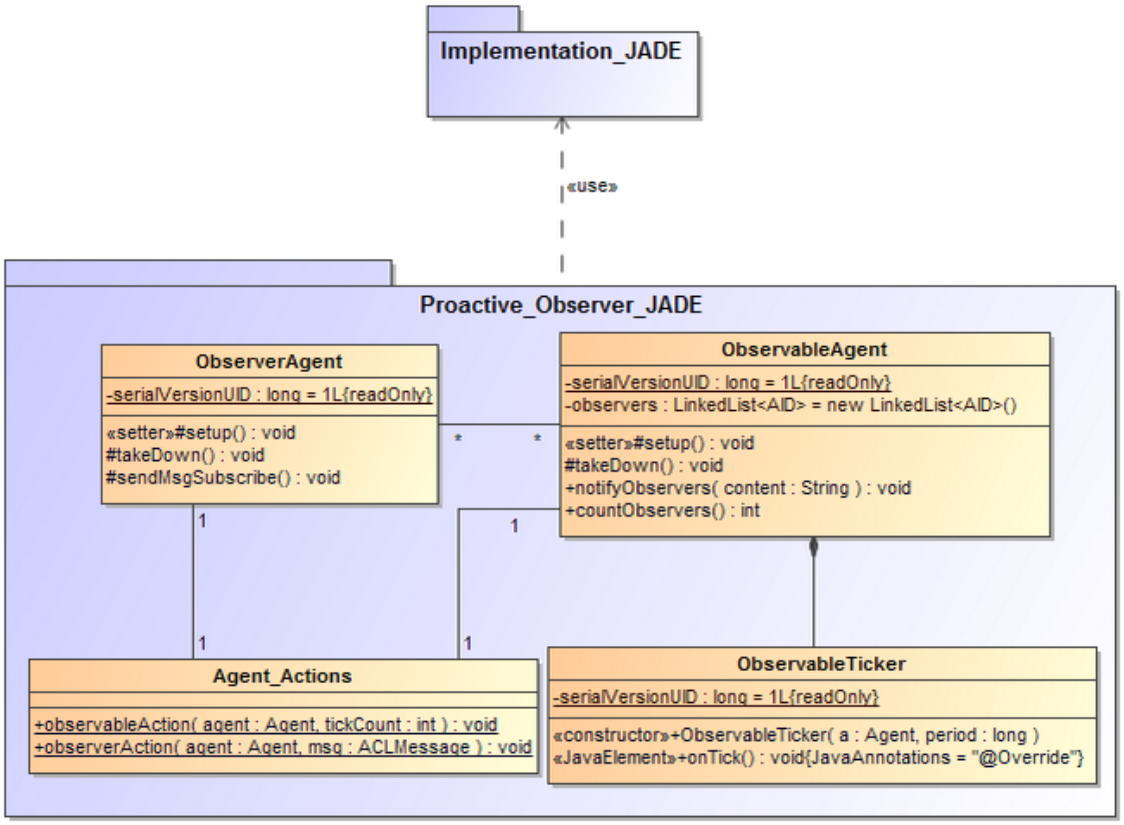



Figura 13: Diagrama de clases del patrón *Proactive Observer_JADE* y su relación con el patrón *Implementation_JADE*.

*ObservableAgent* extiende de *Agent*, escucha los mensajes del tipo *Subscribe* [50] que le llegan de un *Observer*. Cuando un mensaje de este tipo es recibido, se decide si adicionarlo a la lista de *Observers* o no. Si hay algún cambio se notifica a los *Observers* que se encuentren en la lista. La observación del ambiente se realiza usando la implementación del *ObservableTicker*.

*ObserverAgent* extiende de *Agent*, al ejecutarse se subscribe a un Observable y espera el mensaje de respuesta de si fue subscrito o no. Espera por la notificación del *Observable* y realiza alguna acción.

*ObservableTicker* extiende el funcionamiento de *"TickerBehaviour"* de JADE. Brinda la posibilidad de extender el método *"onTick".* Este método se encarga de verificar sistemáticamente si hubo algún cambio en el entorno.

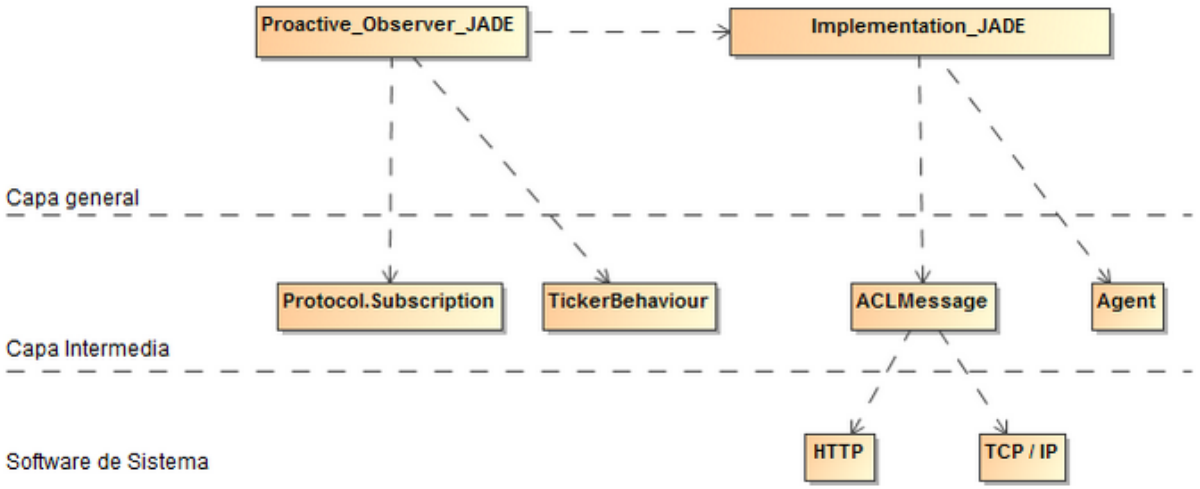



Figura 14: Diagrama en capas de la relación entre las clases.

La Figura 14 describe el modelo en capas que muestra cómo se relacionan las clases. En la capa general están los paquetes que contienen las clases de los patrones *Implementation_JADE* y *Proactive Observer_JADE*. En la capa intermedia se encuentran las clases de la plataforma JADE más utilizadas por los patrones de implementación desarrollados. La capa de software de sistemas tiene los protocolos de comunicación que a nivel del sistema se utilizan para la plataforma JADE.

El funcionamiento de las entidades que se ejecutan en el patrón *Proactive Observer_JADE* puede entenderse mejor en el diagrama de secuencia de la Figura 15. Se indican las diferentes llamadas a las funciones de los agentes para que se vea la interacción entre ellos.

La entidad *ObservableAgent* llama al constructor de *ObservableTicker*, para que así se pueda censar el ambiente cada cierto tiempo. *ObserverAgent* envía un mensaje del tipo *Subscribe* a *ObservableAgent* para subscribirse a él y espera la respuesta del mismo. *ObservableTicker* llama al método "*onTick*" para censar el entorno y ver si ha ocurrido algún cambio. Si hay cambios entonces notifica a

*ObservableAgent* para que envíe un mensaje a los observadores de su lista, los que a su vez actualizan su estado.

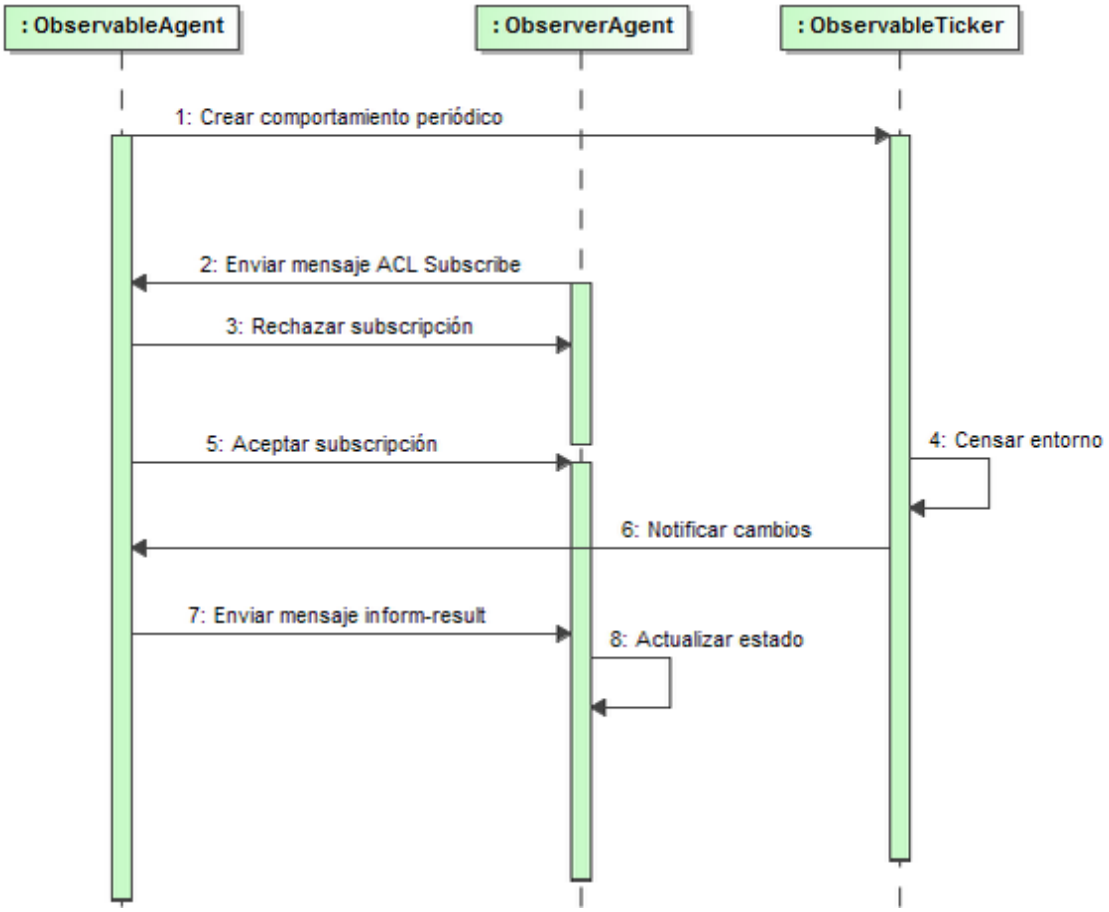



Figura 15: Diagrama de secuencia del funcionamiento del patrón *Proactive Observer_JADE*.

De acuerdo a estas características se implementó un agente *Observable* que al iniciarse espera un mensaje de subscripción de los agentes *Observers* y ejecuta un comportamiento "*Ticker*" que censa el ambiente para informar algún cambio ocurrido. Cuando inicia el *Observer*, él conoce quien debe ser su *Observable*, y le envía un mensaje de subscripción. Es importante notar que este patrón de implementación no sólo respeta el lenguaje FIPA-ACL [50] para la comunicación entre los mensajes, sino que la interacción sigue las indicaciones del protocolo de interacción Subscribe definido también por FIPA [54].

El ciclo de ejecución del patrón sigue los pasos explicados anteriormente y se muestran en la Figura 16, que es una vista gráfica de los agentes ejecutándose en un diagrama propio de la plataforma JADE.

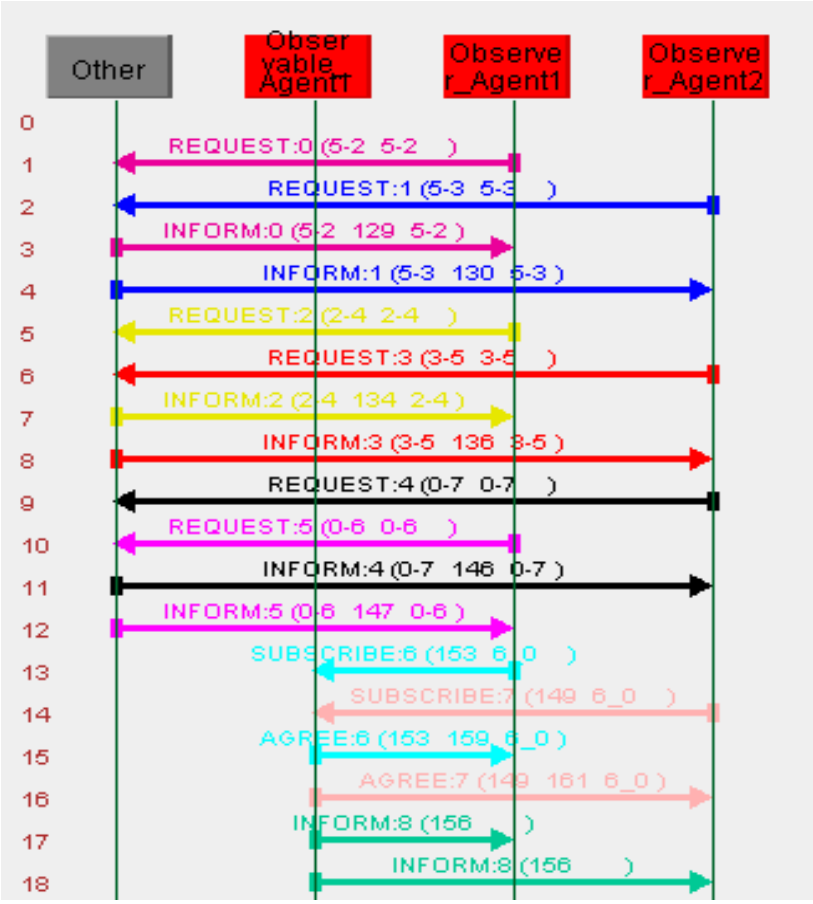



Figura 16: Agentes en la plataforma JADE con el patrón *Proactive Observer_JADE*.

La operación “Guardar trazas de la comunicación entre los agentes” se realiza para guardar ficheros XES [78] con la información de la comunicación entre los agentes. La información de la comunicación puede ser utilizada por los desarrolladores para desarrollar Minería de Procesos [2, 88]. El desarrollador puede guardar o no las trazas de la comunicación ya que esto es una prestación que permite el patrón.

Las clases que permiten guardar trazas de la comunicación entre los agentes están contenidas en un paquete que se encarga de gestionar dicha operación, cuyo diagrama de clases se puede consultar en el anexo 3.

**Consecuencias:** A partir del uso del patrón se garantiza que las entidades *Observer* reciban una notificación encontrada por la entidad *Observable* y puedan tomar decisiones o realizar una acción. Con el uso del patrón se logra además el funcionamiento de los agentes con un despliegue distribuido.

**Clasificación horizontal**: Estructural, de comportamiento y social.

**Clasificación vertical**: Implementación del agente.

### 2.5 Actividades de prueba

Con el objetivo de evaluar el futuro comportamiento del SMA a partir de su modelado de requisitos en i* y de obtener información que permita desarrollar un estudio de la comunicación y el comportamiento de los agentes creados con los patrones propuesto, se desarrollaron dos actividades de prueba.

En la Figura 17 se muestra una particularización del Modelo-V donde se reflejan las actividades de prueba que se proponen a partir de las propuestas durante el ciclo de vida del software.

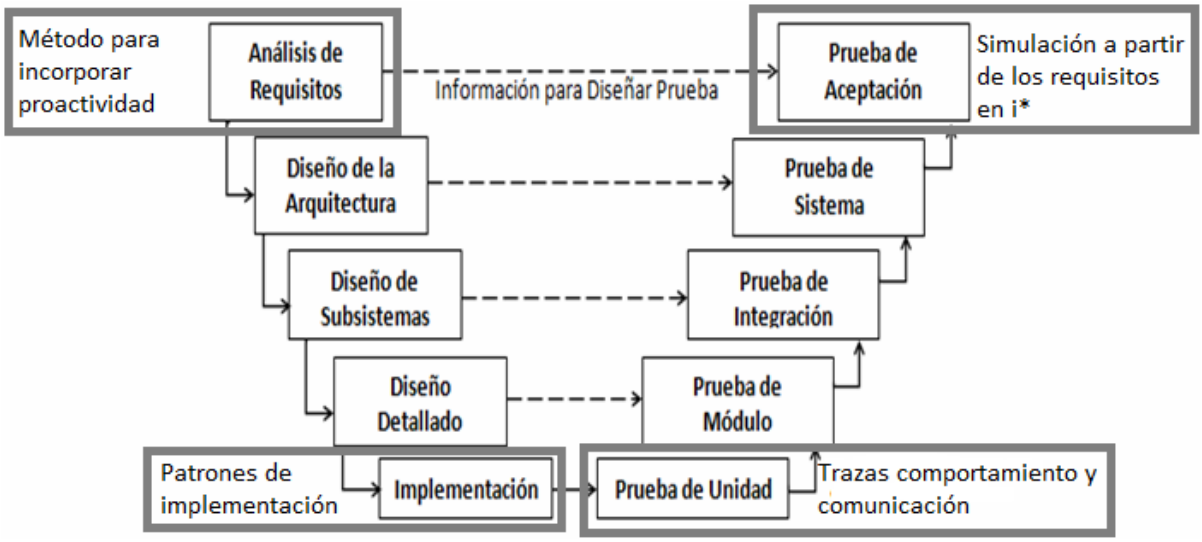



Figura 17: Modelo-V adaptado a la propuesta de incorporación de proactividad.

Como se muestra en la fase de Análisis de requisitos, se obtendrá información para desarrollar una simulación del sistema a desarrollar a partir del modelo de requisitos en el lenguaje i*. En la fase de implementación se obtendrán ficheros con formato XES que pueden ser utilizados para desarrollar Minería de Procesos,

con el objetivo de estudiar el comportamiento y la comunicación entre los agentes creados.

### 2.5.1 Simulación basada en agentes a partir de los requisitos en i*

Para desarrollar la simulación basada en agentes a partir de los requisitos plasmados en i* se formalizó una entrada de variables partiendo del fichero generado por la herramienta Taom4E que soporta los modelos i* [17].

Para la entrada de variables se consideraron los actores, las metas fuertes (hardgoals), las metas suaves (softgoals), los planes y los recursos como elementos.

A partir de la captura de requisitos en i* de un SMA en la herramienta Taom4E, se crea un modelo del cual se puede ver un ejemplo en la Figura 18. En la propuesta desarrollada cada Actor, Rol, Agente o Posición que se encuentre en la etapa de captura de requisitos modelada, se convierte en un tipo de agente o agente de la simulación, manteniendo sus respectivas relaciones.

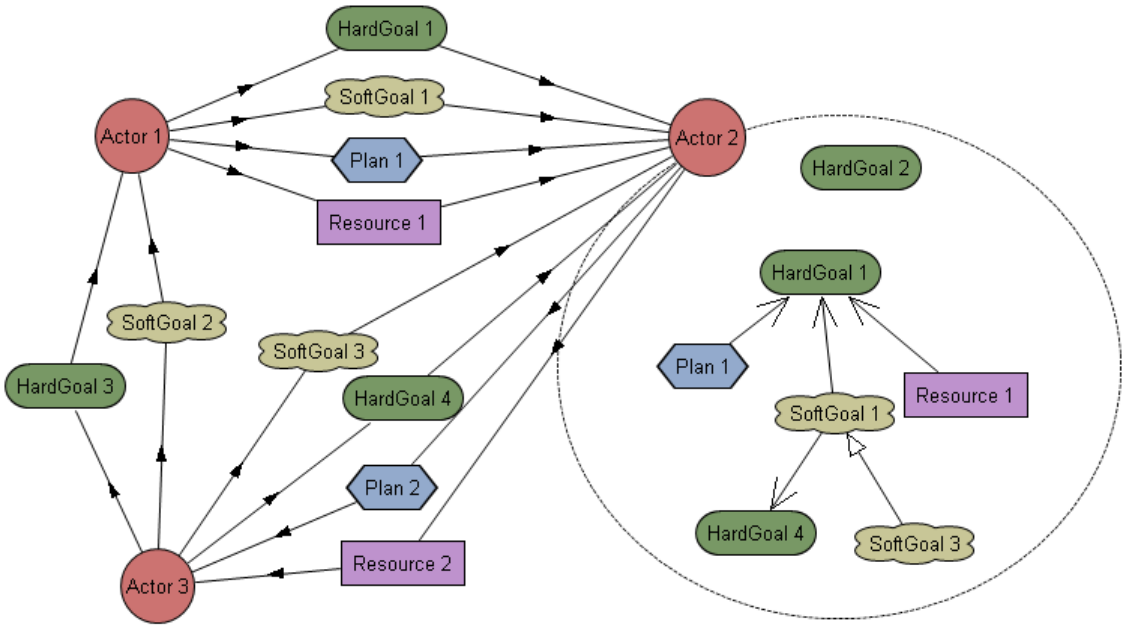



Figura 18: Modelo de la captura de requisitos de un SMA en i*.

Durante la simulación los agentes pueden transitar por cuatro estados. Estos estados son: “inactivo”, “esperando respuesta”, “procesando respuesta” y

“procesando”. En el estado de “inactivo” se encuentran los agentes que participan en la simulación y se encuentran inactivos por no estar realizando ninguna tarea en ese momento. Por el estado “procesando” pasan los agentes que contienen al menos, un elemento interno que no presenta relaciones con ningún elemento de una dependencia, y a su vez, no es un elemento de una dependencia. En este estado, estos elementos se deben estar procesando en todo momento, mientras que el agente responsable del mismo no se encuentre en otro estado.

Por otro lado, en los estados de “esperando respuesta” y “procesando respuesta”, se encontrarán los agentes que piden un elemento y los que deben responder a dichos pedidos, respectivamente. Una vez que se satisface el pedido, los agentes involucrados pasarán al estado inactivo, mientras no tengan pedidos pendientes por realizar. Cada tipo de agente tiene una cola de pedidos que se activa cuando la cantidad de pedidos a un tipo de agente supera la cantidad de instancias que hay en la simulación de este tipo de agente.

## 2.5 Conclusiones Parciales

Al concluir el presente capítulo sobre la solución propuesta se pueden realizar las siguientes conclusiones:

- Se pueden encontrar requisitos proactivos en el modelo de los requisitos tempranos plasmado en i*. Los patrones *Hardgoal why Dependency* y *Resource why Dependency* a partir de un problema en el modelo o en un subconjunto de modelo de los requisitos tempranos en i* dan una solución para delegar intenciones al software en el modelo de requisitos tardíos de i* y que este pueda ser desarrollado con un comportamiento proactivo.
- El patrón de implementación *Implementation_JADE* puede facilitar la construcción de un sistema multi-agente que tome como base la plataforma JADE. Es la base para el desarrollo del patrón de implementación *Proactive Observer_JADE*. Además puede facilitar la hibridación entre objetos y agentes.

- El patrón de implementación *Proactive Observer_JADE* puede facilitar la implementación de comportamientos proactivos a partir de la observación periódica del ambiente.
- El patrón de implementación *Proactive Observer_JADE* permite una opción de guardar fichero XES que pueden ser utilizados en el estudio del comportamiento del sistema multi-agente con técnicas de Minería de Procesos como parte de las pruebas de unidad.
- A partir del modelado de requisitos en i* se puede obtener una simulación basada en agentes que sirve de base a las prueba de aceptación.

# *Capítulo 3*

# *Estudio de Casos: desarrollo de sistemas informáticos proactivos*

# *Capítulo 3: Estudio de Casos: desarrollo de sistemas informáticos proactivos*

## 3.1 Introducción

En este capítulo se presentan dos casos: el desarrollo de un observatorio tecnológico proactivo y la concepción de metaheurísticas proactivas. Ambos se basan en el estudio de casos como método para la investigación. Es importante destacar que se escogen dos casos con características diferentes. En un observatorio tecnológico es sabido que es recomendable tener proactividad para ayudar a sus usuarios, pero en las metaheurísticas no se han desarrollado trabajos que expongan los beneficios de la proactividad.

## 3.2 Estudio de Casos

El estudio de casos constituye hoy un método importante y reconocido para la investigación [42, 174]. En el caso particular de la Informática se ha usado y defendido su uso [34, 42, 109] en situaciones que son difíciles de reproducir en un experimento de laboratorio en que es fácilmente aislable y controlable el contexto. Aquí, es importante profundizar en una realidad, en su complejidad de relaciones y variables participantes de un modo más cualitativo, más que intentar resumir la evidencia en un simple dato. 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 [34, 174]. También permiten que los mismos autores sean parte de la investigación [34].

No obstante, es importante aclarar que la generalización que se realiza con el método de estudio de casos no se basa en la generalización estadística que ve cada unidad como una muestra, sino que se basa en la generalización analítica que ve cada unidad como un experimento [174] para comprobar o no una teoría previamente enunciada. Además de los casos que se presentan en este capítulo,

en [48, 153, 160] se muestran otros ejemplos de evidencias que corroboran los resultados obtenidos del estudio.

Los dos casos fueron escogidos por su relevancia, siguiendo las recomendaciones de [174]. En ambos, la unidad de análisis es el desarrollo de un sistema informático. Un caso trata sobre el desarrollo de un observatorio tecnológico con comportamiento proactivo utilizando JADE. Este puede considerarse como un caso típico según [174], ya que en él se presentan situaciones propias de muchos sistemas de información, con fuerte interacción con usuarios y que permite identificar situaciones que avalen la generalización analítica. Permite estudiar tanto los patrones de requisitos para detectar requisitos proactivos, como el uso de los patrones de implementación propuestos. El otro caso trata sobre el desarrollo de metaheurísticas con comportamiento proactivo. Este puede clasificarse como un caso extremo según [174] ya que las metaheurísticas son sistemas complejos muy dependientes de las configuraciones fijadas por los usuarios, donde no se ha introducido comportamiento proactivo a pesar de que se han tratado otros aspectos de la orientación a agentes. En este caso el énfasis está en identificar posibilidades de proactividad en diferentes dimensiones, ninguna de las cuales se había modelado así. En este último sentido, este segundo caso también puede verse como revelatorio porque permite enfocarse en aspectos no tratados antes [174].

En ambos, el propósito es la validación de las propuestas presentadas en el capítulo 2, por lo que pueden ser clasificados como explicativos. De las seis estructuras de presentación de los casos [174] 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 [42, 109, 174] los casos se presentan haciendo explícitos algunos elementos importantes de este método.

- Pregunta(s) de estudio y proposiciones: Se aclara la pregunta en qué 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 [174].
- 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.

## 3.3 Caso 1: Observatorio Tecnológico

Se presenta el desarrollo de un Observatorio Tecnológico como caso 1.

### 3.3.1 Preguntas de estudio y proposiciones

¿Cómo los patrones de requisitos para detectar proactividad utilizando los requisitos en i* ayudan a identificar requisitos proactivos?

¿Cómo los patrones de implementación propuestos simplifican el desarrollo de un sistema informático proactivo?

Las proposiciones correspondientes son:

- la utilización del lenguaje i* y los patrones de requisitos para detectar proactividad permiten identificar requisitos proactivos, y

- los patrones de implementación propuestos ayudan a simplificar el desarrollo de un observatorio tecnológico proactivo.

### 3.3.2 Contexto

La unidad de análisis es el proceso de desarrollo de un observatorio tecnológico.

Un observatorio tecnológico (OT) es una herramienta que apoya la vigilancia tecnológica [141], reconoce cambios en el dominio de información que procesa, gestiona y observa, por lo tanto, teniendo en cuenta comportamientos previos, puede avisar con antelación de ciertas variaciones o diferencias en los parámetros que evalúa. Un OT genera un conocimiento con un alto nivel de importancia al ser actual y novedoso, que puede ser utilizado por los receptores que tengan interés en esa información [36].

Según Bouza, un OT captura informaciones externas con el propósito de transformarlas en conocimientos específicos que conduce a sus usuarios a tomar decisiones [116]. Un OT es un sistema de alerta para identificar y recopilar aquellos datos e informaciones que pueden ser fuente de amenaza u oportunidad.

Si se toma como base las definiciones anteriores se puede concluir que un OT mide y procesa elementos concernientes a un tema. Alivia el trabajo de buscar información relevante que tribute a los intereses de los usuarios, gracias a la integración en una herramienta que busca información circunscrita a temas determinados, que provee de informes, resúmenes y alertas, que permitan a los usuarios tomar decisiones.

Como se plantea, un OT funciona principalmente para dar respuesta a los intereses de sus usuarios. Cuando se desea construir un OT es esencial estudiar mecanismos que permitan responder los intereses de los usuarios con la menor intervención humana. La mayoría de los OT operan gracias a las personas que trabajan dándole soporte, buscando, procesando, resumiendo, colocando noticias en los sitios web e informando a los clientes de sus descubrimientos. Un ejemplo lo constituye ESTO (European Science and Technology Observatory) [167], una red de 18 organizaciones europeas con experiencia en el campo científico y

tecnológico, que funciona bajo la Comisión de las Comunidades Europeas, y opera como un instituto virtual. Otros OT con esta característica se dedican al tratamiento de información centrada en dominios específicos, como son: Observatorio Chileno de Ciencia, Tecnología e Innovación (KAWAX) [45], del Plástico [166] y de la Soldadura [139], los cuales tienen un personal extenso que estudia estas ramas del mercado y analizan las noticias que se publican relacionadas con sus temas. Como sus respectivos nombres lo indican, cada OT es una referencia en el tema que procesa, dando a cada sector del mercado información confiable e importante para los usuarios, de forma dinámica, periódica y actualizada. Pero esta información no es personalizada, es para usuarios con metas comunes y no con metas específicas particulares [36].

Según un estudio desarrollado en [47] donde se consultaron un conjunto de expertos en ciencias de la información, todos plantearon la necesidad de proactividad en los sistemas de gestión de información como los observatorios tecnológicos. Los sistemas deben ser capaces de adelantarse a las solicitudes de información de los usuarios, proveyéndoles de información precisa, siguiendo las metas de sus usuarios.

Por tanto, un caso a tener en cuenta la proactividad es en la construcción de un observatorio tecnológico. Sería útil que los OT tengan la capacidad de ser proactivos en cuanto a la búsqueda de información, de estar orientados a metas a partir de las necesidades de sus usuarios. Los observatorios deben utilizar un método claro, riguroso y neutro de alerta temprana para sus usuarios [141]. No se conocen de modelos explícitos de las relaciones entre los actores de un OT, ni de las arquitecturas para lograr su implementación [120].

### 3.3.3 Lógica de análisis

En la lógica del análisis se abordarán las preguntas de estudio, las proposiciones y cómo éstas serán respondidas.

#### 3.3.3.1 ¿Cómo los patrones de requisitos para detectar proactividad utilizando los requisitos en i* ayudan a identificar requisitos proactivos?

En muchos centros de investigación, como es el caso del Complejo de Investigaciones Tecnológicas Integradas (CITI), de la CUJAE se busca información directamente de las fuentes de datos. La Figura 19 expone los requisitos tempranos, modelados con i*, de la forma en que se busca la información en el Complejo de Investigaciones.

El actor Investigador tiene como meta principal “Investigar”, para cumplimentar esta meta depende del actor Fuentes para “Buscar información”. El actor Fuentes a su vez necesita “Conocer los intereses” del Investigador. Para dar a conocer sus intereses, el Investigador cuenta con un perfil, que representa un recurso con las áreas de investigaciones en las que trabaja y las palabras claves relacionadas. Para dirigir y tener mejor perfilados sus intereses, el Investigador depende de un Investigador Experto que le ayuda saber cuáles palabras claves buscar, qué autores, qué fuentes bibliográficas de un área de investigación.

Una de las intenciones del Investigador, representada por una meta suave, es “Mantenerse informado según intereses”, a esta meta contribuye positivamente la meta “Conocer los intereses”. Esta meta suave “Mantenerse informado según intereses” contribuye positivamente a la meta principal del Investigador que es “Investigar”, pero es sólo una intención, ya que de forma automatizada aún no cuenta con los medios para cumplimentar la misma.

Al hacer un análisis del modelo de los requisitos tempranos con i* siguiendo el patrón *Hardgoal why Dependency,* propuesto en el capítulo 2, se puede comprobar que:

- Existe un actor Investigador (que representa un humano) que tiene una meta fuerte “Investigar”, a esta meta fuerte le contribuye positivamente una meta suave “Mantenerse informado según intereses”, dicha meta suave representa una intención. Ambas metas son sólo del actor Investigador, no se derivan de una dependencia.

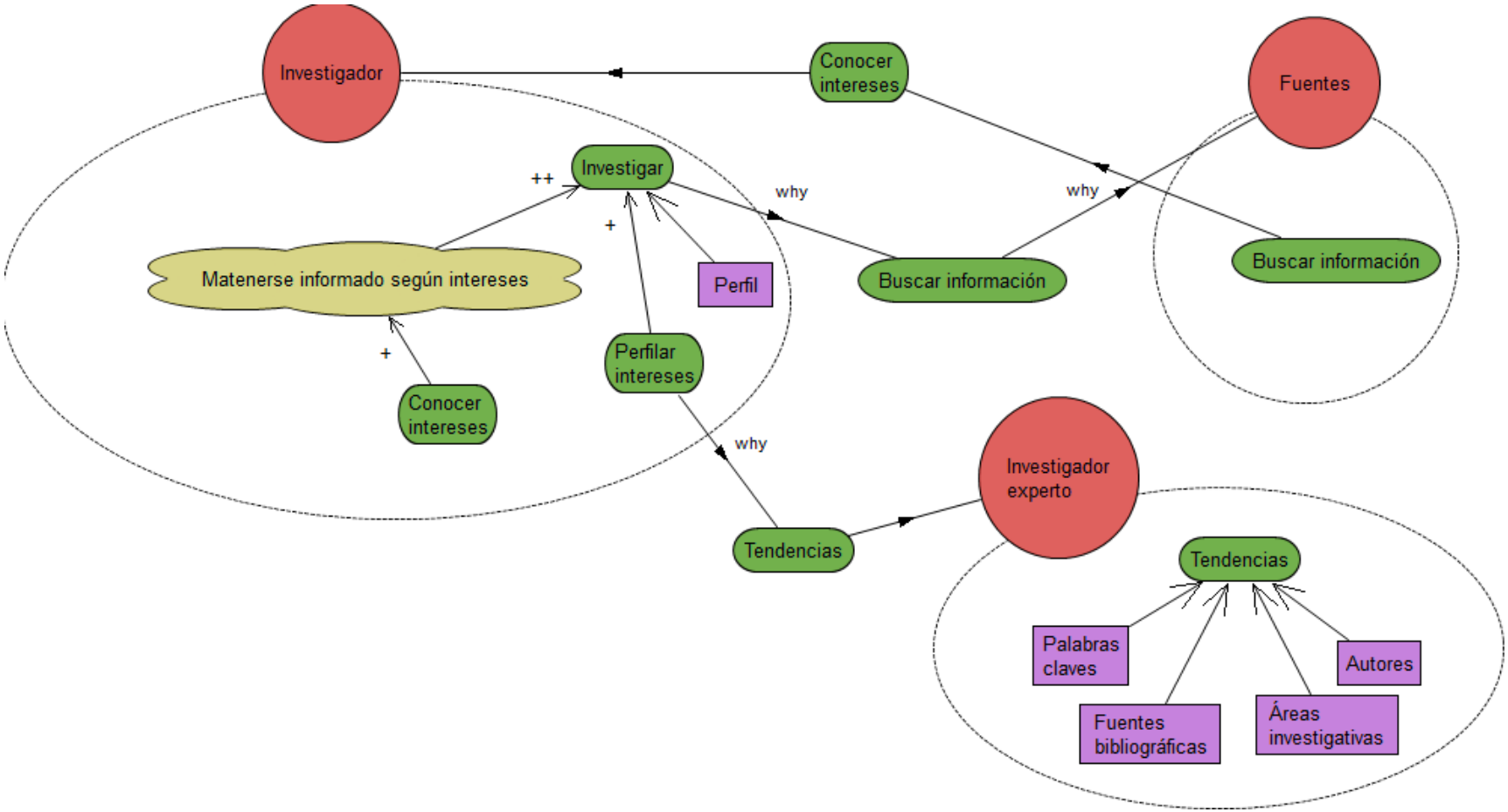



Figura 19: Modelo de los requisitos tempranos con i* de la búsqueda de información en un centro de investigación.

- Para cumplimentar la meta “Investigar” el actor Investigador tiene una dependencia con el actor Fuentes de la meta “Buscar información”.
- El actor Fuentes (que representa una entidad de software) para cumplimentar la meta “Buscar información” depende de la meta “Conocer los intereses” del actor Investigador.
- La meta “Conocer los intereses” contribuye positivamente a la intención o meta suave “Mantenerse informado según intereses”.

Existen además otras dependencias, pero las descritas anteriormente entre el actor Investigador y el actor Fuentes son las que permiten ver la intencionalidad de la relación.

Teniendo en cuenta la solución propuesta por el patrón *Hardgoal why Dependency,* las entidades y sus relaciones en el subconjunto del modelo abordado se reflejarán en el modelo de requisitos tardíos con i* como se expone a continuación:

- El actor Fuentes se transforma en el sistema a desarrollar, en este caso agente Observatorio.
- Al actor (agente) Observatorio se le delega la meta suave “Mantener informado según intereses”, que representa una intención.
- Al actor (agente) Observatorio se le delega además la meta “Conocer intereses”.
- La meta fuerte “Conocer intereses” se transforma en plan “Conocer intereses”, que es un medio para cumplimentar la meta suave “Mantener informado según intereses”.
- Se refleja que la meta suave “Mantener informado según intereses” contribuye positivamente a la meta “Buscar información” que se realiza para cumplimentar la meta “Investigar”.

En la Figura 20 se muestra el modelo obtenido de los requisitos tardíos con i* del sistema del OT que se desea desarrollar, aplicando el patrón *Hardgoal why Dependency*.

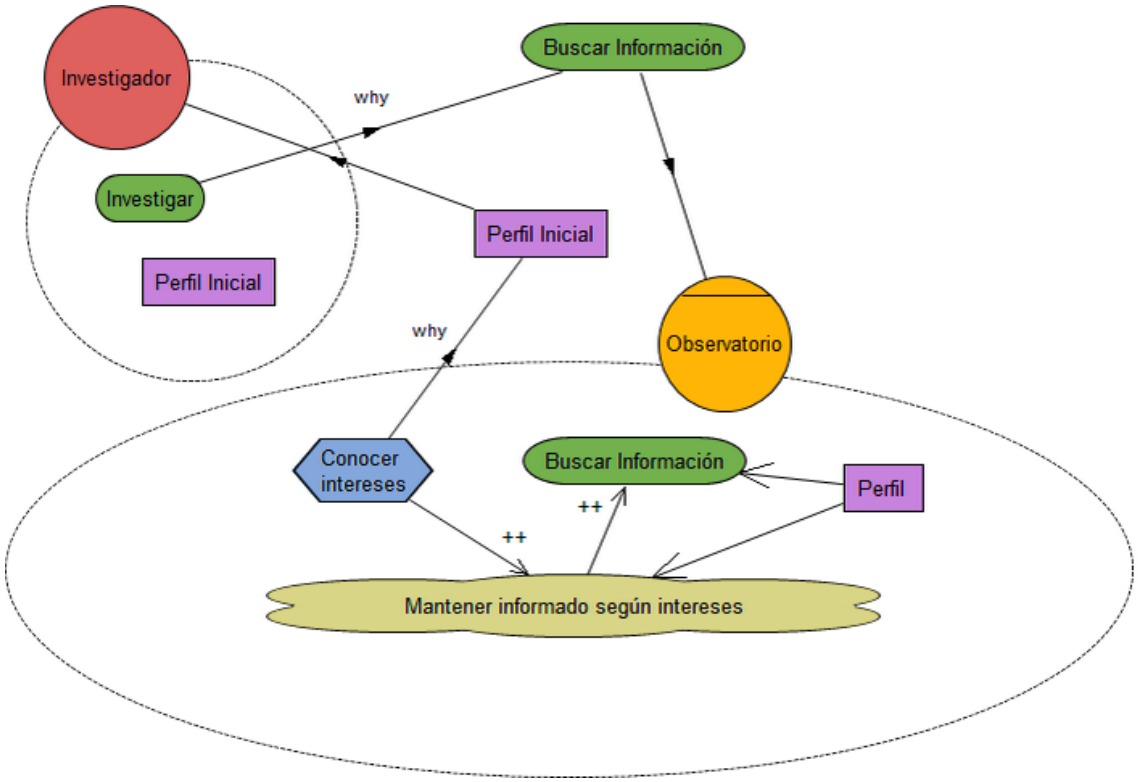



Figura 20: Modelo de los requisitos tardíos con i* del sistema observatorio tecnológico que se va a desarrollar.

### 3.3.3.2 ¿Cómo los patrones de implementación propuestos simplifican el desarrollo de un sistema informático proactivo?

El sistema informático proactivo que se desea desarrollar es el Observatorio Tecnológico. Al hacer un análisis de la tecnología que debe soportar la construcción del OT, se valoró utilizar la tecnología de agentes. Los agentes permiten incorporar proactividad y desarrollar un sistema distribuido que facilita el desempeño de un sistema de gestión de información como lo es un Observatorio Tecnológico [26, 47, 111, 151].

Al tener en cuenta que el observatorio tecnológico iba a estar sustentado por un sistema multi-agente, se hizo un despliegue de funcionalidades por los agentes

(entidades de software que van a formar parte del OT). Siguiendo las recomendaciones de [26, 142], cada usuario del OT debe estar representado por un entidad agente, que sería su cara al sistema, en este caso el Agente Personal.

Utilizando el lenguaje i* se puede plasmar los actores que conformarán el sistema Observatorio Tecnológico y la relación entre ellos.

En la Figura 21 se muestra un modelo de requisitos tardíos de todas las entidades que conformaran el OT soportado por un sistema multi-agente. El OT está compuesto por una capa cliente y una capa para el sistema multi-agente. En la capa cliente está la Interfaz Gráfica. El SMA está compuesto por un repositorio y cuatro tipos de agentes: agente personal (AP), agente de confianza (AC), agente analista (AA) y agente fuente de datos (AFD). El SMA además consta de tres módulos, orientados a objetos, en los que se apoyan el AP y AFD. Estos son: LuceneIndex, URLDownloader y Search Docs.

Cuando el usuario ejecuta la Interfaz Gráfica, se levanta un AP que lo representará dentro del SMA. El despliegue concebido del SMA implica un AP por cada usuario del sistema. Cada AP está a cargo de la gestión de los recursos de información disponibles, se tiene en cuenta el perfil y el comportamiento del usuario ante la Interfaz Gráfica. El perfil del usuario se guarda en el repositorio, lo que permite que cuando el usuario se conecte desde cualquier computadora, obtenga todos los datos que le corresponden, una vez que se autentique en el sistema. El perfil de usuario le permite al SMA atender y responder a las necesidades independientes de cada uno.

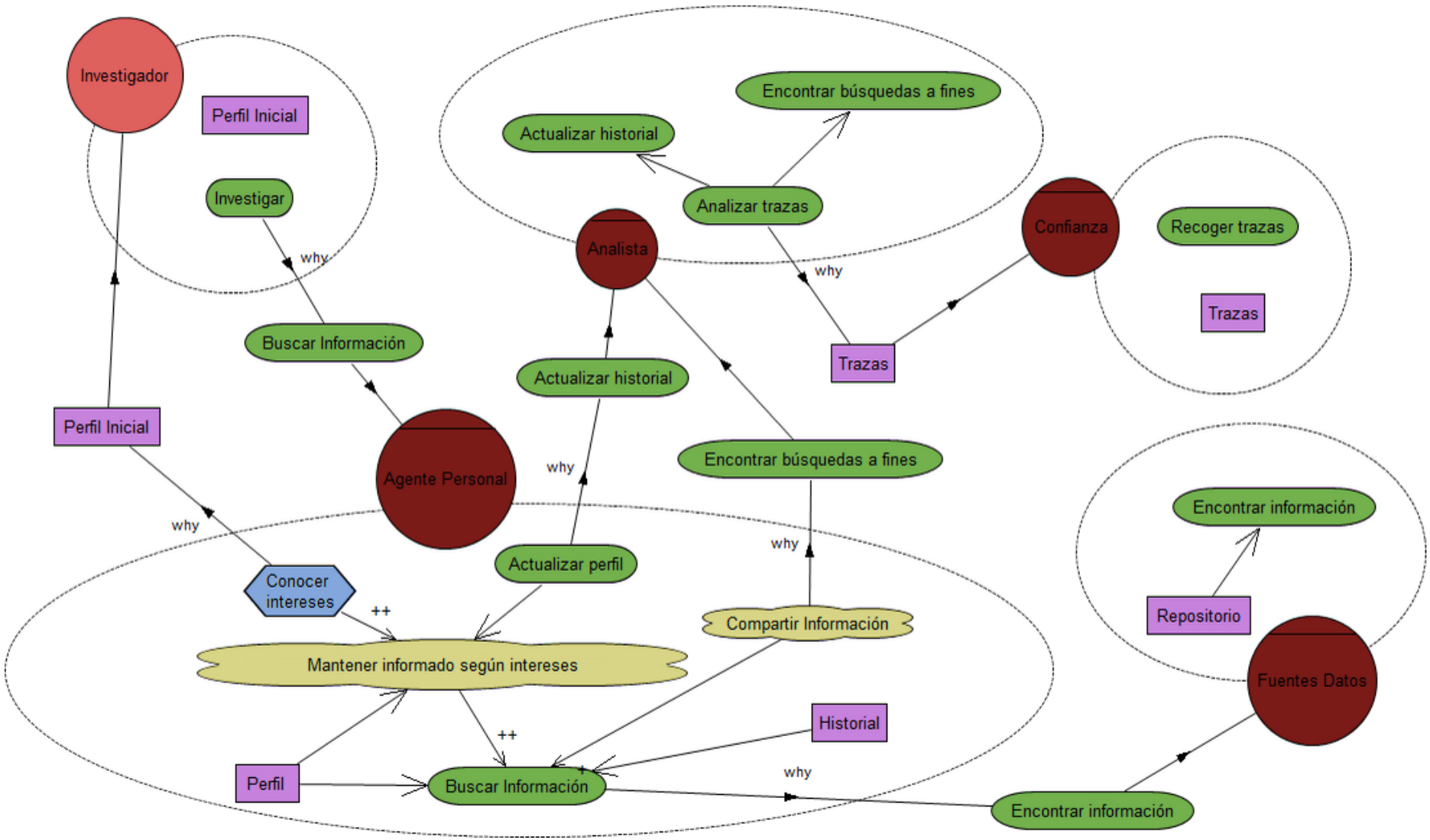



Figura 21: Modelo de los requisitos tardíos con i* del sistema observatorio tecnológico con una arquitectura basada en agentes.

En sentido general, cada AP se dedica a buscar, descargar e indexar la información procedente de fuentes abiertas externas que el usuario solicita consultar, recuperar y compartir con otros AP, teniendo en cuenta las políticas de colaboración establecidas por el usuario y su perfil, el cual es traducido en metas del AP. Cada AFD está alerta para atender los pedidos del AP para buscar y descargar del FTP[6] que monitorea. Existe un AFD que analiza el repositorio que forma parte del OT. El AFD indexa la información que está en el FTP y crea índices más fáciles de consultar.

El AC escucha los mensajes que se envían entre los agentes del SMA y guarda una traza de esta comunicación. Las trazas las utiliza el AA para determinar, mediante procesos de Minería de Datos, por ejemplo, cuáles usuarios son especialistas en un tema, o cuáles están trabajando en temas similares. El AA además puede preguntar al AC la cantidad de recursos de información de una temática que ha enviado cada AP del SMA o algo más general, como la cantidad de mensajes relacionados con un tema que ha enviado cada AP, ya sean de recuperación o de contribución de contenido.

Desde etapas tempranas, se puede tener un acercamiento al funcionamiento de esta arquitectura, enfocada a probar el sistema a partir de la simulación, usando la propuesta presentada en la sección 2.5.1. El anexo 4 muestra la simulación del OT con distintas configuraciones, así como el análisis de los resultados.

En el desarrollo de un sistema multi-agente que da soporte a un observatorio tecnológico con características proactivas se presentan dos problemas que se enuncian a continuación.

**Problemas**

1. El primero está relacionado con la dificultad del trabajo con JADE de forma general, el número de líneas de código a escribir para hacer la gestión de los

[6] FTP: Siglas en inglés de File Transfer Protocol

agentes, los contenedores, la plataforma, envío y recepción de mensajes ACL, etc. Esto se repite en cada tipo de agente que componen el Observatorio Tecnológico.

2. Otro problema se encuentra cuando un especialista necesita una información que es relevante para su trabajo y debe esperar a que en la planificación de su Agente Personal se realice la búsqueda de recursos. Esto conlleva a que el especialista debe esperar un tiempo para recibir resultados de la búsqueda.

Se quiere que de forma periódica el AP haga un reconocimiento del entorno para encontrar información nueva y relevante para su usuario, respondiendo a la intención de “Mantener informado según intereses”. Primero, se debe comunicar con otros AP y si estos no dan una respuesta satisfactoria debe pasar a tramitar sus pedidos con el Agente Fuente de Datos. Cuando hay un cambio, se encuentra algo nuevo, el AP debe enviar un correo electrónico a su usuario con los resultados.

**Soluciones**

En la implementación del sistema se utilizó el patrón *Implementation_JADE* y el patrón *Proactive Observer_JADE*. A continuación se explica cómo fueron utilizados cada uno.

1. El sistema del Observatorio utiliza las clases de apoyo para ejecutar la plataforma JADE brindadas por el patrón *Implementation_JADE*, para inicializar los contenedores y los agentes, para comunicarse con los agentes del servidor JADE (ams y df), además envía y recibe los mensajes ACL entre agentes. El programador realiza todas estas funciones de forma sencilla y flexible con menos esfuerzo. Todos los agentes desarrollados en el Observatorio utilizan el patrón *Implementation_JADE.* En el anexo 5 se puede ver cuántas líneas de códigos son necesarias para crear un agente con las funcionalidades básicas sin el patrón (172) y utilizando el patrón (24).

2. Se implementó un agente **FuentedeDatos_Agent** que extiende las funcionalidades de la clase *ObservableAgent*, que por las características de este sistema la función que efectúa es gestionar un sitio (repositorio). También se

implementan los agentes **Personal_Agent** que extienden de la clase *ObserverAgent*, la cual en este sistema atiende a los especialistas. Ambas clases se vinculan por medio del patrón *Proactive Observer_JADE*.

De forma natural el agente personal solicita la búsqueda de documentos a los agentes fuentes de datos del sistema, de forma que los últimos realizan búsquedas en sus correspondientes sitios y devuelven una lista de aquellos documentos que cumplan con las palabras pedidas por los usuarios.

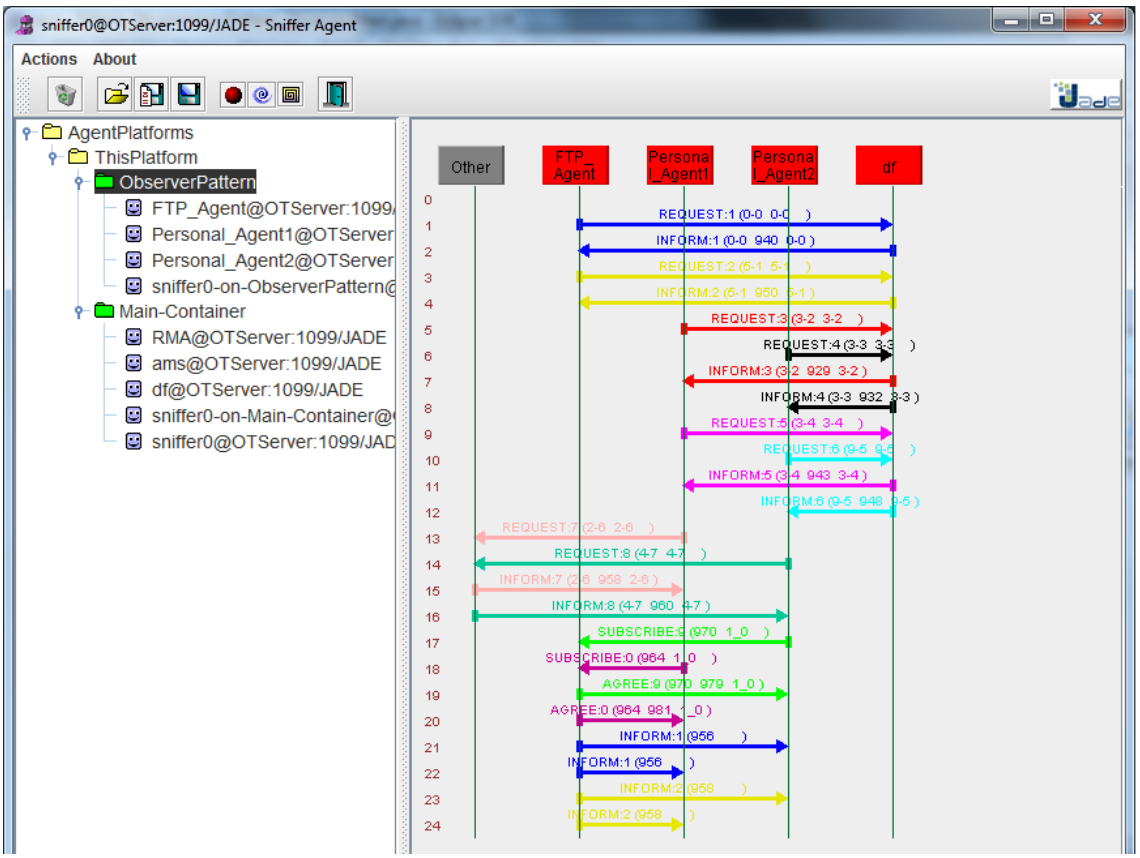


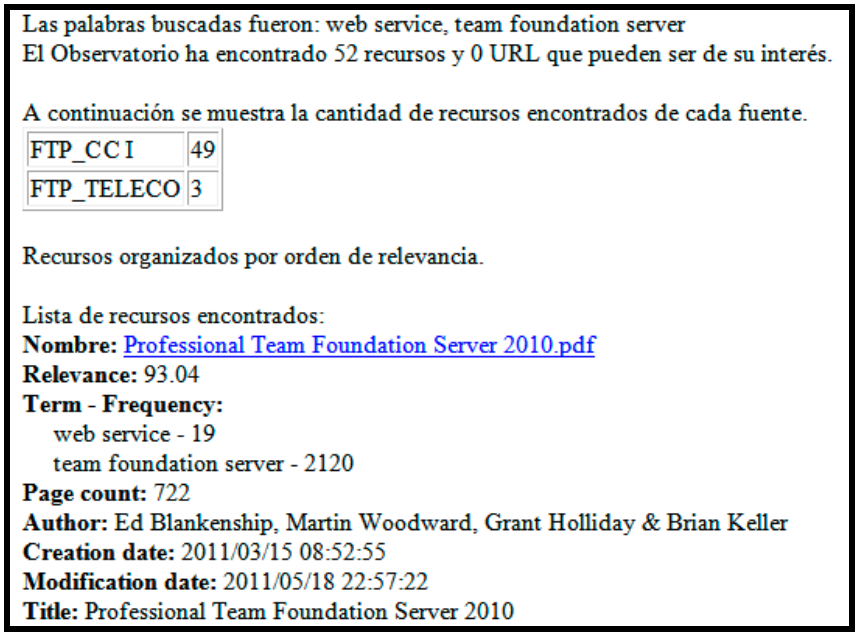

Figura 22: Flujo de mensajes con el patrón *Proactive Observer_JADE* en el Observatorio.

Al utilizar el patrón *Proactive Observer_JADE,* una persona puede desear subscribirse a un sitio, consiguiendo que el agente Fuente de Dato que gestiona ese sitio pueda enviar resultados en el momento en que los encuentra al Agente Personal que atiende a ese usuario. Este último entonces envía un correo con los

resultados encontrados antes de tener que realizar una búsqueda que viene condicionada por la planificación en su ciclo.

Los Agentes Personales asumen el rol de *Observer* y el Agente Fuente de Datos el de *Observable*. En la Figura 22 se muestra como los Agentes Personales se subscriben al Agente Fuente de Datos (FTP_Agent) de la misma forma en que fue explicado el patrón *Proactive Observer_JADE* anteriormente.

En la Figura 23 se muestra un ejemplo del correo enviado por el Agente Personal al especialista. Todo la información relevante a un usuario se obtiene de forma proactiva, con solo decir sus intereses. El agente personal que representa al usuario con sus intereses, utilizando el patrón *Proactive Observer_JADE*, se mantiene buscando cada cambio, en los sitios que se escogen. Cuando hay un cambio, el usuario recibe un correo con lo nuevo encontrado en los sitios o lo que ha socializado otro agente personal.



Las palabras buscadas fueron: web service, team foundation server
El Observatorio ha encontrado 52 recursos y 0 URL que pueden ser de su interés.

A continuación se muestra la cantidad de recursos encontrados de cada fuente.

| | |
|---|---|
| FTP_CC I | 49 |
| FTP_TELECO | 3 |

Recursos organizados por orden de relevancia.

Lista de recursos encontrados:
**Nombre:** Professional Team Foundation Server 2010.pdf
**Relevance:** 93.04
**Term - Frequency:**
web service - 19
team foundation server - 2120
**Page count:** 722
**Author:** Ed Blankenship, Martin Woodward, Grant Holliday & Brian Keller
**Creation date:** 2011/03/15 08:52:55
**Modification date:** 2011/05/18 22:57:22
**Title:** Professional Team Foundation Server 2010

Figura 23: Correo electrónico del resultado de la ejecución de los patrones *Implementation_JADE* y el patrón *Proactive Observer_JADE* en el Observatorio.

### 3.3.4 Discusión

Como se puede observar, en la captura de los requisitos tempranos de cómo se efectúa la investigación en un Complejo de Investigaciones se dan las condiciones

de dependencia entre dos actores que denotan intencionalidad. En este caso el Investigador desea mantenerse informado según sus intereses con la meta de investigar. Pero para investigar necesita de las fuentes de datos externas para encontrar la información. Las fuente de datos no pueden buscar información a ciegas, ellas necesitan tener los intereses del Investigador. Al darse estas dependencias se puede delegar en el sistema tecnológico a construir la meta de mantener informado al usuario, para lo cual necesita de un plan para conocer los intereses del Investigador. De esta forma el sistema observatorio buscará la información y mantendrá informado al Investigador sin necesidad de que él mismo pida la información. Se puede entonces concluir que con la utilización de los patrones de requisitos para detectar proactividad utilizando los modelos de requisitos en i* se puede identificar requisitos proactivos en un sistema informático que se desarrollará.

Por otro lado, se desarrolló un sistema multi-agente para dar soporte al observatorio tecnológico que tuviera un comportamiento proactivo. Para la construcción de cada tipo de agente, su comunicación, sus cambios de comportamiento, se utilizó el patrón de implementación *Implementation_JADE* que facilitó toda la construcción del sistema multi-agente. Para lograr un comportamiento proactivo se utilizó el patrón *Proactive Observer_JADE.* Utilizando este patrón, los agentes personales que representan a cada usuario se subscriben a diferentes fuentes de datos y observan los cambios en el entorno para entregar información relevante al Investigador sin que este la pida. Lo único que necesita el Agente Personal es saber los intereses del Investigador.

En el anexo 6 se ejemplifica cómo el sistema tiene un comportamiento proactivo en una prueba piloto. Este ejemplo muestra cómo a partir de un número de peticiones, que representan los intereses de los usuarios, se generan un número de mensajes proactivos como el mostrado en la Figura 23.

Debido a que la presentación del caso y su discusión de ha basado en las relaciones entre las entidades descritas y no a detalles concretos de la semántica del ejemplo, es posible notar que el caso puede servir como ejemplo de uso de las

propuestas en situaciones similares en que se den las mismas dependencias. De esta manera el estudio de caso ha sido muy útil como herramienta de investigación. Algunos resultados presentados aquí fueron publicados en [120, 121].

## 3.4 Caso 2: Metaheurísticas Proactivas

Se presenta la concepción de metaheurísticas proactivas como caso 2.

### 3.4.1 Pregunta de estudio y proposiciones

¿Cómo los patrones de requisitos para detectar proactividad utilizando los requisitos en i* permiten identificar requisitos proactivos para desarrollar metaheurísticas proactivas?

La proposición en este caso es que la utilización del lenguaje i* y los patrones de requisitos para detectar proactividad permiten desarrollar metaheurísticas proactivas.

### 3.4.2 Contexto

La unidad de análisis es el desarrollo de metaheurísticas proactivas.

Las metaheurísticas son bien conocidas como métodos aproximados de optimización de funciones en dominios o espacios de soluciones grandes, que no garantizan encontrar el óptimo, pero que usualmente obtienen soluciones suficientemente buenas en un tiempo razonable [162]. Algunos ejemplos de metaheurísticas son el recocido simulado, la búsqueda tabú y los algoritmos evolutivos. En los años recientes han surgido muchas nuevas metaheurísticas, variantes y combinaciones de otras [20, 22, 30, 41, 152, 162]. La característica esencial de las metaheurísticas, que le dan su carácter general, es no asumir ninguna restricción sobre el problema a tratar, ni sobre la función objetivo a optimizar, solo necesitan que se defina una forma de representar y evaluar las soluciones, así como operadores para construir una solución inicial, y para transformar las soluciones en otras nuevas.

Saber cuándo conviene usar cada metaheurística es un problema abierto. El Teorema No Free Lunch (NFL) establece que todas las metaheurísticas se comportan igual en promedio, y si una metaheurística es mejor que otra para unos problemas, debe esperarse que existan otros problemas donde ocurra lo contrario [170].

Debido a la gran cantidad de metaheurísticas, este trabajo se enfocará en las metaheurísticas basadas en un punto o S-Metaheurísticas [162]. Todas estas S-Metaheurísticas se basan en mantener una solución como referencia (en ocasiones llamada solución actual), a partir de la cual se generan nuevas soluciones por la aplicación consecutiva de operadores. Las S-Metaheurísticas brindan la mejor solución encontrada durante el proceso de optimización. La forma de trabajo de las metaheurísticas depende en gran medida del balance entre dos factores: la explotación (o intensificación) y la exploración (o diversificación) [21, 162].

La búsqueda aleatoria (RS, por las siglas en inglés de *Random Search*) está en un extremo de la exploración, esto se debe a que cada solución nueva es generada sin tener en cuenta las soluciones generadas anteriormente. Por otro lado, la búsqueda local (LS, por las siglas en inglés de *Local Search*), también llamada escalador de colinas, está en el otro extremo ya que genera nuevas soluciones que son la modificación de las mejores soluciones anteriores. Una nueva solución es aceptada únicamente como referencia para generar nuevas soluciones si es mejor que las soluciones encontradas anteriormente. Este criterio de aceptación puede llevar a converger en un óptimo local (no global), donde la optimización se estanca.

Los óptimos locales son consecuencia de los operadores, que son los que definen la vecindad [94], y del criterio de aceptación. Se han diseñado varias S-Metaheurísticas para superar este problema, relajando el criterio de aceptación, y considerando algunas soluciones peores como nuevas referencias. Por ejemplo, en Camino Aleatorio (RW, por las siglas en inglés de *Random Walk*) cada nueva solución (peor o mejor) se acepta como referencia. Otras S-Metaheurísticas, como

el Recocido Simulado (SA, por las siglas en inglés de *Simulated Annealing*), la Aceptación por Umbral (TA, por las siglas en inglés de *Threshold Accepting*), *Record-to-Record Travel* (RRT) y Algoritmo del Gran Diluvio (GDA, por las siglas en inglés de *Great Deluge Algorithm*) utilizan un criterio moderado de aceptación. Estas últimas metaheurísticas aceptan algunas soluciones peores teniendo en cuenta la calidad de la nueva solución, y algunos otros aspectos y parámetros. Por ejemplo, RRT acepta soluciones que sean peores que la actual si la diferencia entre la nueva y la mejor solución encontrada no es mayor que un parámetro llamado Desviación. Por su parte, TA acepta una solución peor que la actual si la diferencia entre ambas no excede un parámetro llamado umbral T. En el caso de GDA, acepta una solución nueva, independientemente de que sea mejor o peor que la actual, pues la aceptación solo depende de que esta tenga una evaluación superior a un parámetro llamado nivel de agua WL, que se incremente sistemáticamente adicionando un valor llamado lluvia R. Cada uno de estos parámetros afecta el comportamiento de los algoritmos. Puede verse que el parámetro Desviación afecta directamente la realización del comportamiento de RRT, porque controla el balance entre la explotación y exploración. Por ejemplo, RRT con un valor muy alto de desviación es similar a RW.

En contraste con la modificación del criterio de aceptación, un enfoque diferente para evitar los óptimos locales es modificar la vecindad.

Este enfoque de modificar la vecindad es utilizado por Búsqueda con Vecindad Variable (VNS, por las siglas en inglés de *Variable Neighborhood Search*). Como la vecindad cambia, una solución que es un óptimo local en una vecindad no es necesariamente un óptimo local en otra vecindad. La idea subyacente en VNS es que la mejor solución al final de la búsqueda puede ser un óptimo global porque ha sido un óptimo local en muchas estructuras de vecindad. Los operadores y los criterios para cambiarlos afectan la realización de la VNS. Existen disponibles algunas pautas generales para ajustar todas las S-Metaheurísticas [18, 162], pero los buenos valores dependen de los problemas, los operadores utilizados, y del estado actual del proceso de optimización.

Se han propuesto muchas métricas para entender y predecir la actuación de la metaheurísticas, por ejemplo la correlación entre la distancia y la evaluación llamada FDC (del inglés *Fitness Distance Correlation*) [94]. FDC mide la correlación entre la distancia en representación de cada solución y el óptimo, respecto a la calidad en evaluación (*fitness*) de cada solución. Cuando la correlación es cercana a -1, significa que mientras las soluciones están a menor distancia del óptimo, su evaluación es mayor. Si FDC=-1 el problema puede ser fácil, pero si FDC= 1 el problema puede ser difícil. Como FDC necesita los valores de toda la solución, es normal utilizarlo para estudiar el comportamiento de las metaheurísticas en algunos tipos de problemas. FDC depende del operador para establecer la distancia. Por ejemplo, las soluciones 000 y 101 están a la distancia de dos, en términos de mutar un bit cada vez, pero están a una distancia de uno en términos de mutar dos bits.

En general, el ajuste de parámetros y de vecindades en el desarrollo de las metaheurísticas es desarrollado por los humanos (optimizadores). Ellos imaginan lo que hace falta y no delegan las metas ni los recursos para estos ajustes a la metaheurística.

No se han encontrado trabajos que modelen explícitamente las metas en el trabajo con metaheurísticas, ni utilizando i*, ni de otra manera, y tampoco que se haya intentado dotarlas de un comportamiento proactivo en función de lograr sus propias metas. Tampoco se han encontrado referencias a la utilización de la filosofía de agentes para ajustar los parámetros de las metaheurísticas para lograr ciertas metas. Aunque hay numerosos trabajos sobre mecanismos para el ajuste de parámetros en metaheurísticas [18, 19, 30, 138, 162], en estos trabajos no se han definido explícitamente las metas que la metaheurísticas deben intentar lograr, ni se ha visto la evolución de la metaheurística como un ambiente que puede ser observado y sobre el que las metaheurísticas pueden comportarse como un agente, que puede actuar en él a través de sus parámetros, para lograr las metas del optimizador. Las metaheurísticas se han utilizado para optimizar un sistema de agentes (por ejemplo [73], lo cual constituye un interés recíproco al de este trabajo.

### 3.4.3 Lógica de análisis

Se verá la pregunta de estudio: ¿Cómo los patrones de requisitos para detectar proactividad utilizando los requisitos en i* permiten identificar requisitos proactivos para desarrollar metaheurísticas proactivas?

Se expondrá cómo del análisis del modelado de los requisitos se pueden obtener requisitos proactivos. Se presentará además cómo se pueden desarrollar metaheurísticas proactivas.

En la Figura 24 se muestran los requisitos tempranos modelados en i* de una S-Metaheurística. En este modelo hay tres actores estratégicos: S-Metaheurística, Función y Optimizador.

El Optimizador (ente humano) tiene una meta “Optimizar”, para cumplimentar esta meta depende del actor S-Metaheurística (software) para “Encontrar mejor solución”. La S-Metaheurística depende de la Función (software) para saber la evaluación de cada solución. La S-Metaheurística puede realizar planes, como son: “Generar solución inicial”, “Generar nueva solución”, entre otros. La meta suave o intención del Optimizador es “No estancarse” en óptimos locales.

Al hacer un análisis del modelo de los requisitos tempranos con i* siguiendo el patrón *Hardgoal why Dependency* propuesto en el capítulo 2, se puede constatar que:

- Existe un actor Optimizador (que representa un humano) que tiene una meta fuerte “Optimizar”, a esta meta fuerte le contribuye positivamente una meta suave “No estancarse”, dicha meta suave representa una intención. Ambas metas son sólo del actor Optimizador, no se derivan de una dependencia.
- Para cumplimentar la meta “Optimizar” el actor Optimizador tiene una dependencia con el actor S-Metaheurística de una meta “Encontrar mejor solución”.
- El actor S-Metaheurística (que representa una entidad de software), para cumplimentar la meta “Encontrar mejor solución”, tiene que ejecutar un plan

"Actualizar referencias". El plan "Actualizar referencias" necesita los parámetros representados por el recurso "Parámetros". Para obtener estos parámetros el actor S-Metaheurística tiene una dependencia del actor Optimizador de la meta "Ajustar parámetros".

- La meta "Ajustar parámetros" contribuye positivamente a la intención o meta suave "No estancarse".

También al hacer un análisis del modelo de los requisitos tempranos con i* siguiendo el patrón *Resource why Dependency* propuesto en el capítulo 2, se puede comprobar que:

- Existe un actor Optimizador (que representa un humano) que tiene una meta fuerte "Optimizar", a esta meta fuerte le contribuye positivamente una meta suave "No estancarse", dicha meta suave representa una intención. Ambas metas son sólo del actor Optimizador, no se derivan de una dependencia.
- Para cumplimentar la meta "Optimizar" el actor Optimizador tiene una dependencia con el actor S-Metaheurística de una meta "Encontrar mejor solución".
- El actor S-Metaheurística (que representa una entidad de software) para cumplimentar la meta "Encontrar mejor solución" tiene que ejecutar un plan "Generar nueva solución". El plan "Generar nueva solución" para cumplimentarse tiene una dependencia del actor Optimizador para el recurso "Operador".
- El "Operador" es un medio para cumplimentar la intención o meta suave "No estancarse".

Existen además otras dependencias, pero las dependencias descritas anteriormente entre el actor Optimizador y el actor S-Metaheurística son las que permiten ver la intencionalidad de la relación.

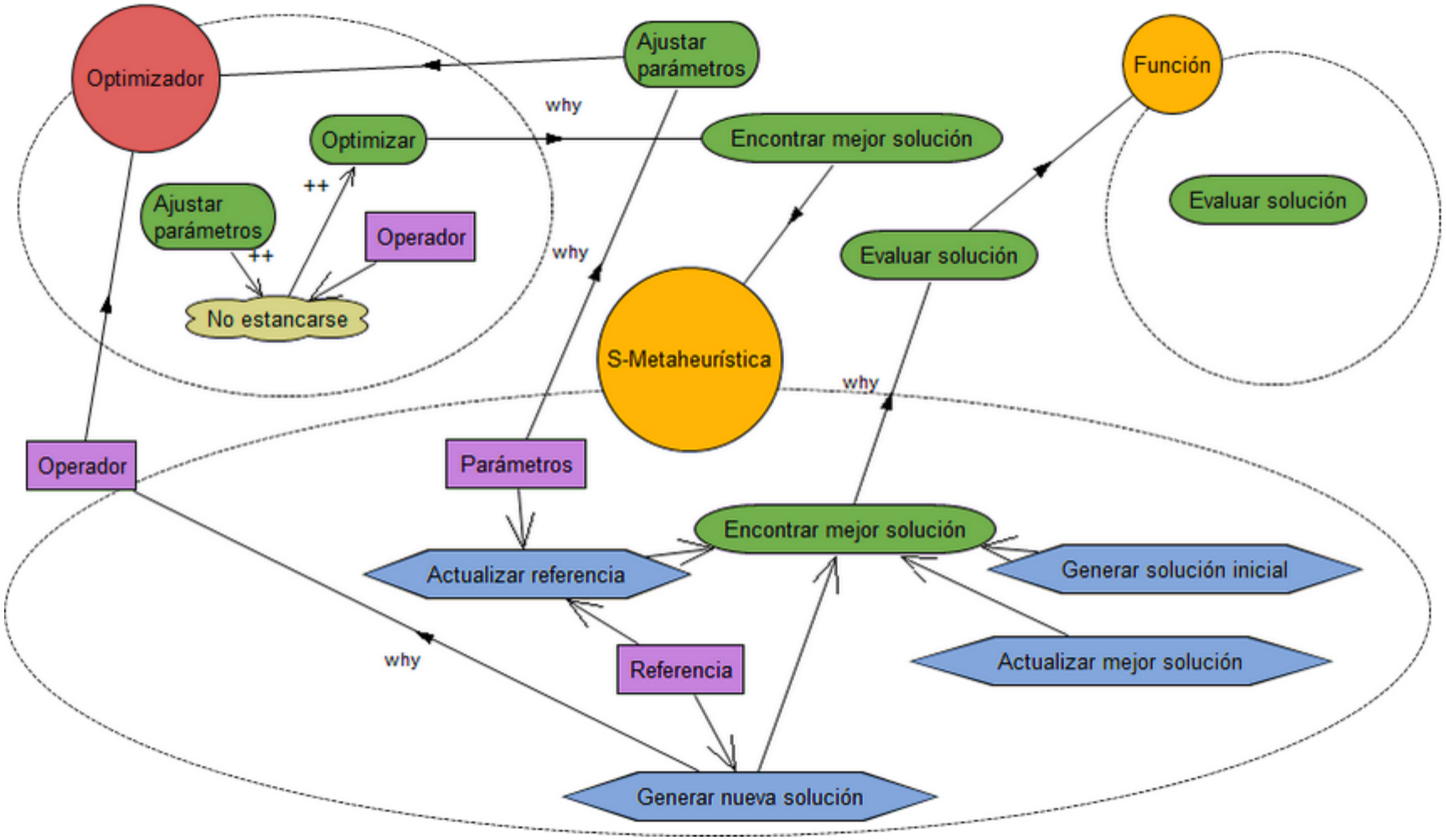



Figura 24: Modelo de los requisitos tempranos con i* de la realización una S- Metaheurística.

Teniendo en cuenta la solución propuesta por el patrón *Hardgoal why Dependency,* las entidades y sus relaciones de una parte del subconjunto del modelo abordado se reflejarán en el modelo de requisitos tardíos con i* como se expone a continuación:

- El actor S-Metaheurística se transforma en el sistema a desarrollar (aumentar funcionalidades) en este caso agente S-Metaheurística.
- Al actor (agente) S-Metaheurística se le delega la meta suave “No estancarse”, que representa una intención.
- Al actor (agente) S-Metaheurística se le delega además la meta “Ajustar parámetros”.
- La meta fuerte “Ajustar parámetros” se transforma en plan “Ajustar parámetros”, que es un medio para cumplimentar la meta suave “No estancarse”.
- Se refleja que la meta suave “No estancarse” contribuye positivamente a la meta “Encontrar mejor solución” que se realiza para cumplimentar la meta “Optimizar”.

Teniendo en cuenta la solución propuesta por el patrón *Resource why Dependency* las entidades y sus relaciones, la otra parte del subconjunto del modelo abordado se reflejarán en el modelo de requisitos tardíos con i* como se expone a continuación:

- Al actor S-Metaheurística se le delega además el recurso “Operador”. En este caso el recurso “Operador” seguirá siendo un medio para cumplimentar el plan “Generar nueva solución”. También será un medio para cumplimentar plan “Ajustar vecindad”, que es un medio para cumplimentar la meta suave “No estancarse”.

La Figura 25 se muestra el modelo de los requisitos tardíos en i* de la S-Metaheurística proactiva a desarrollarse.

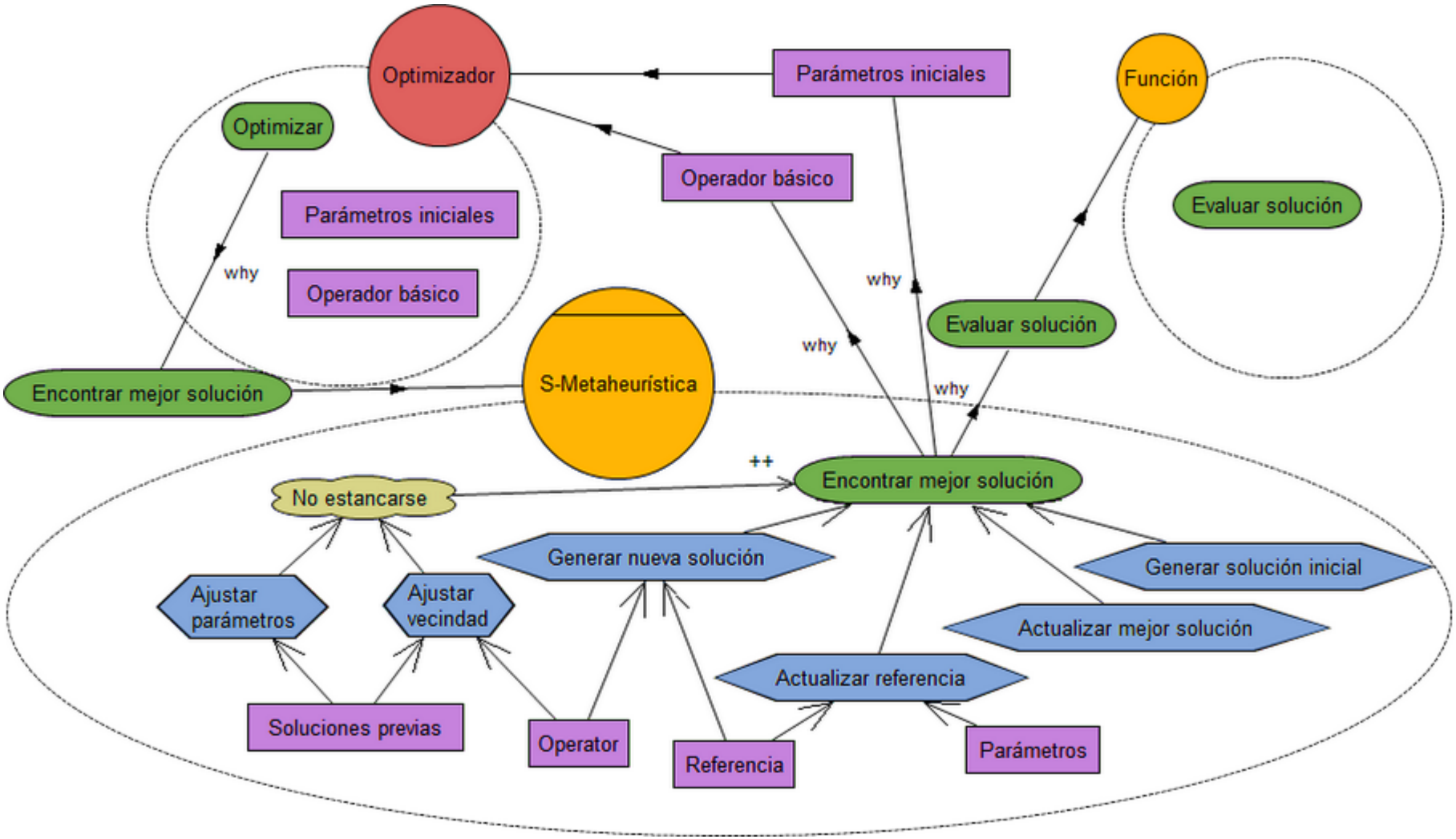



Figura 25: Modelo de los requisitos tardíos con i* de la S-Metaheurística proactiva a desarrollarse.

### 3.4.3.1 Plan para el ajuste proactivo de los parámetros

La captura de requisitos con i* permitió identificar que cada S-Metaheurística dispone de acciones (ajustes de los parámetros) para intentar satisfacer la meta suave “No estancarse” en un óptimo local.

El ambiente puede utilizarse de muchas maneras, pero puede verse que la aceptación de nuevas soluciones va a depender, en principio, del valor de la evaluación de la función objetivo en la nueva solución. Para garantizar que en el futuro inmediato las metaheurísticas acepten al menos una solución de las generadas y que no se queden estancadas, los valores de los parámetros deben ser ajustados.

Considerando una ventana de tiempo representada por TW[7] soluciones generadas, se pueden ordenar éstas según el valor de su evaluación de manera descendente, de manera que $E_1$ sería la mejor solución encontrada y $E_{TW}$ sería la peor. Si se quiere que según la calidad de la función se acepte la mejor en una ventana de tiempo, los parámetros de las metaheurísticas pueden ajustarse para que el punto de corte (PC) entre la que se acepte y las que no se acepten quede entre los valores $E_1$ y $E_2$, por ejemplo tomando el promedio de ambos, es decir $PC= (E_1 + E_2)/2$. El valor PC constituye la concreción de la meta de aceptación de una solución (y por tanto, de no quedarse estancada) que se le asignó a la metaheurística. Como no se puede predecir las soluciones que se obtendrán en la próxima ventana de tiempo, el punto de corte se establece a partir de la ventana de tiempo actual. De esta manera se pretende tener un comportamiento proactivo previendo lo que sucederá en la próxima ventana de tiempo. Por tanto, pueden adaptarse los parámetros de cada metaheurística para lograr esto, de la manera siguiente.

- Para TA, el parámetro umbral T puede ajustarse usando la expresión $T= \max (E_r – PC, 0)$, siendo $E_r$ el valor de evaluación de la referencia actual. Esto se hace para evitar que el valor del umbral sea negativo.

[7] TW: cantidad de soluciones generadas en una ventana de tiempo.

- Para GDA, el parámetro nivel del agua WL puede ajustarse usando la expresión WL= PC.
- Para RRT, el parámetro desviación D puede ajustarse usando la expresión D= $E_m$ – PC, siendo $E_m$ el valor de evaluación de la mejor solución encontrada.

Con las expresiones anteriores, las tres metaheurísticas comentadas pueden ajustar sus parámetros de manera proactiva para evitar estancarse en un óptimo local aceptando al menos una solución entre las TW soluciones. Puede notarse que de esta manera los parámetros T, WL y D no son entradas de las metaheurísticas, sino que son recursos con los que cuenta la metaheurística para lograr las metas de aceptación.

#### 3.4.3.2 Plan para el ajuste proactivo de la vecindad

Entendiendo la vecindad de una solución como el conjunto de soluciones que se pueden obtener a partir de ella aplicando un operador definido, se puede entender como estructura de vecindad un grafo donde cada nodo es una solución y cada arco entre ellas implica la existencia de un operador que permite transitar de una solución a otra [94, 164]. Cambiar el operador significa cambiar la vecindad. Esto puede extrapolarse a un número mayor, siendo las soluciones vecinas de una solución aquellas que se obtienen de aplicar el operador definido N veces. Es importante aclarar que la aplicación repetida del operador solo implica un cambio en la estructura de la vecindad y no mayor cantidad de evaluaciones de la función objetivo. La aplicación repetida del operador produce vecindades nuevas, sin necesidad de obligar al usuario a definir diferentes operadores.

No es simple ajustar el valor de N por el usuario, ni saber cuál de esos operadores es el más conveniente. Más aún, este parámetro no es una meta en sí mismo, sino que es un recurso del que se puede dotar a la metaheurística para explorar el espacio de búsqueda de una manera más eficiente. Esta exploración más eficiente sí puede ser una meta a delegar en la metaheurística. Para lograr satisfacerla, la metaheurística dispone de recursos (parámetros, historia de soluciones generadas, etc.) para intentar lograrla. La historia de la evolución es un ambiente

en que la metaheurística actúa, y en el que trata de lograr encontrar la solución con el mejor valor posible en el tiempo del que dispone. En base a eso, puede ajustar proactivamente la forma de explorar para hacer la búsqueda más eficiente.

La historia de la evolución brinda información sobre las soluciones anteriormente generadas. Como las nuevas soluciones se obtienen por modificaciones (mutaciones) esto brinda información sobre la estructura de las vecindades. Con estas soluciones puede tenerse una aproximación de cómo es la zona más cercana del espacio de búsqueda. Esta información puede servir para estimar lo que sucederá en un futuro cercano.

**Evaluación de vecindades con FDC**

El ambiente (historia) puede servir para analizar cuál estructura de vecindad es más conveniente, y la métrica FDC es un buen estimador de la dificultad de esta para la búsqueda local LS. La definición original de FDC exige un conocimiento total del espacio de búsqueda lo cual no es posible siempre. Sin embargo, puede obtenerse una aproximación de FDC usando una muestra del espacio de búsqueda, y se puede tomar la mejor solución de esa muestra como un pseudo-óptimo a los efectos del cálculo de FDC.

En el caso de LS (Búsqueda Local o escalado de colinas), cada nueva solución es generada a partir de la mejor solución (o de la más reciente de ellas si hay varias mejores). De esta manera, con un análisis retrospectivo de las últimas TW soluciones generadas, se puede obtener una serie de valores de distancia al pseudo-óptimo según el orden en que fueron generadas. Como también se cuenta con el valor de evaluación de estas soluciones se puede calcular un valor aproximado de FDC. La Tabla 3 muestra un ejemplo de una secuencia de soluciones generadas por LS. La columna E muestra la evaluación obtenida para cada solución. Como el valor más alto es 30, este se usa como pseudo-óptimo para el cálculo de FDC. La columna Cambio muestra una “x” cuando LS modificó la referencia, de esta forma todas las soluciones generadas sin cambio de referencia están a distancia 1 de ella. La columna N=1 muestra la distancia en cantidad de veces que hay que aplicar el operador básico que se esté usando

entre cada solución y el pseudo-óptimo. La soluciones 25 y 12 fueron generadas a partir de 30, y están a distancia 1. La solución 28 también está a distancia 1 de 30 porque 30 se generó a partir de 28. Transitivamente, como 28 se generó a partir de 26, entonces 26 está a distancia 1 de 28, y a distancia 2 de 30. De igual modo, 14 está a distancia 1 de 26, a 2 de 28 y a 3 de 30. Como 10, 13 y 11 están a distancia 1 de 14, están a distancia 4 de 30. Con estos 10 pares de datos se puede calcular el valor de FDC (fila FDC).

Tabla 3: Secuencia de soluciones.

| E | Cambio | Distancia al pseudo-óptimo 30 | | |
|---|---|---|---|---|
| | | N=1 | N=2 | N=3 |
| 10 | x | 4 | 2 | - |
| 14 | x | 3 | - | 1 |
| 13 | | 4 | 2 | - |
| 11 | | 4 | 2 | - |
| 26 | x | 2 | 1 | - |
| 23 | | 3 | - | 1 |
| 28 | x | 1 | - | - |
| 30 | x | 0 | 0 | 0 |
| 25 | | 1 | - | - |
| 12 | | 1 | - | - |
| FDC | | -0.73 | -0.96 | -0.83 |

Con esta misma muestra, se puede calcular FDC para el operador doble (columna N=2, aplicación dos veces del operador básico). Las soluciones que están a una distancia par del óptimo en la columna N=1 pueden alcanzar el óptimo en la mitad de los pasos, y las que están a distancia impar no pueden llegar al óptimo, y no se sabe la distancia para ellos (indicado con “-” en la Tabla 3). Con esta serie de cinco pares de datos se puede calcular igualmente el valor de FDC para la estructura de vecindad con N=2. De manera análoga puede obtenerse para N=3.

Estas estructuras de vecindad relevantes para calcular FDC para distintos valores de N se muestran en la Figura 26. Según N crece aparecen menos datos para estimar FDC para la estructura de vecindad definida para ese N. Por ejemplo, como no hay ningún valor de 5 en la columna N=1, no es posible estimar FDC

para el operador de mutar 5 veces (N=5). La tabla 2 muestra un ejemplo de soluciones generadas en una ventana de tiempo pequeña (TW=10 soluciones), lo cual hace imposible estimar FDC para N=5 o más. Si TW fuera mayor, entonces sí podría hacerse.

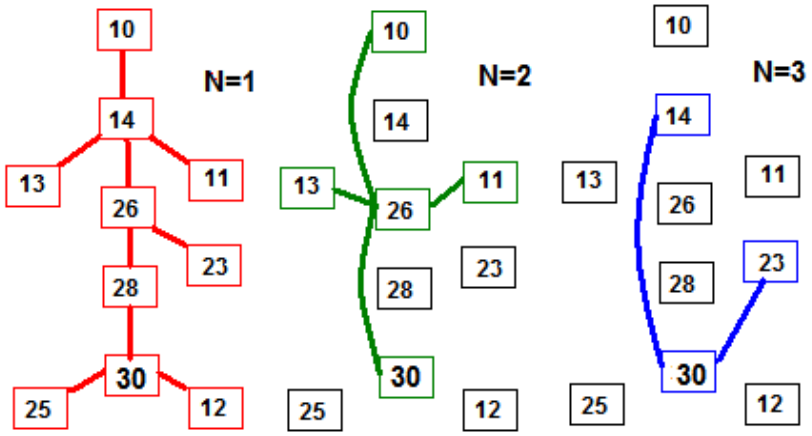



Figura 26: Vecindades respecto al pseudo-óptimo.

**Selección proactiva de la vecindad**

Analizando los valores de FDC en la Tabla 3 y considerando que LS funciona bien cuando FDC está cerca de -1, puede verse que N=2 es el que proporciona una estructura de vecindad más favorable con $FDC_{N=2}$ = - 0.96. En base a esto, durante una próxima ventana de tiempo, la soluciones deberían generarse aplicando dos veces el operador básico de mutación (N=2), y por tanto explotando la estructura de vecindad que parece más conveniente. En general, en cada paso se escoge el valor de N que implica un mejor FDC.

Esta decisión muestra cómo puede delegarse en la metaheurística la meta de explorar la vecindad de una manera más eficiente, para lo cual cuenta con los resultados de las últimas soluciones generadas que funcionan como un ambiente que puede ser analizado y utilizado proactivamente para lograr mejores resultados.

Es importante también analizar lo que ocurre cuando se generan soluciones con N=2. Siguiendo un razonamiento similar al que se hizo en la sección anterior respecto a la Tabla 3, ahora se pueden estimar los valores de FDC para N=2, y

sus múltiplos. Por ejemplo, después de una ventana de tiempo usando N=2, se tiene un nuevo pseudo-óptimo. Luego, las soluciones que sirvieron para generar ese pseudo-óptimo y las que se generaron directamente a partir de él estarían a distancia 1 con N=2. Con esos valores a distancia 1 usando N=2 se puede recalcular $FDC_{N=2}$. El análisis que se hizo para distancia 2 en la Tabla 3 ahora se estaría refiriendo a N=4, y así sucesivamente.

Al pasar una ventana de tiempo, se puede recalcular FDC para diferentes valores de N, aunque no es posible hacerlo para todos en cada ventana de tiempo. Para mantener disponibles el máximo de estructuras de vecindades en cada momento, se mantiene una lista global de valores de FDC para cada N. Cuando pasa una ventana de tiempo TW, los nuevos valores de FDC calculados actualizan los anteriores, promediando el valor anterior y el nuevo calculado, y con esto se actualiza la lista de valores de FDC. Luego de esta actualización, se utiliza como operador de vecindad el correspondiente al valor de N que mejor FDC tenga, lo cual implica la aplicación de N repeticiones del operador de mutación básico.

#### 3.4.3.3 S-Metaheurísticas Proactivas

Como se explicó antes, cada tipo de decisión proactiva puede utilizarse para definir diferentes tipos de S-Metaheurísticas Proactivas. Algunos de ellos son ejemplos de ajuste proactivo de vecindad, otras son ejemplos de ajuste proactivo de parámetros:

- Aceptación por umbral proactiva (PTA): TA con ajuste proactivo del parámetro Umbral.
- Algoritmo del gran diluvio proactivo (PGDA): GDA con ajuste proactivo del parámetro Nivel del Agua.
- RTT proactivo (PRRT): RRT con ajuste proactivo del parámetro Desviación.
- Búsqueda local proactiva basada en FDC (PLS): LS con ajuste proactivo de vecindad (operador) basado en FDC. Como en LS, este método acepta nuevas soluciones si estas son iguales o mejores que las previas.

#### 3.4.4.4 Resultados Experimentales

**Funciones de bloques**

Debido a la gran cantidad de problemas posibles, es necesario acotar siempre el marco experimental a un grupo de problemas que permita validar el interés y la pertinencia de la propuesta. En esta tesis se presentan los resultados para 28 funciones basadas en bloques sobre cadenas binarias, que han sido ampliamente estudiadas [32, 74, 94, 168].

En las funciones basadas en bloques, se toman un grupo de bits consecutivos (bloque), se cuenta la cantidad de "1" presentes en el bloque, y a esta cantidad se le aplican distintas funciones base con distintos grados de dificultad, sumando luego el valor obtenido para cada bloque. En la Figura 27 se muestran algunos ejemplos de funciones base (de tamaño 4) que dada la cantidad de bits correctos (con valor 1) devuelven distintos valores. Es usual trabajar con problemas definidos sobre cadenas binarias de 100 bits, con 25 bloques de longitud 4 cada uno, y con un solo óptimo ubicado en la solución donde todos los bits son iguales a 1, con evaluación de 100. El espacio de soluciones es grande ($2^{100}$= 1,27 * $10^{30}$) y al haber un solo óptimo global conocido (los 100 bits en 1) se puede estudiar el comportamiento de los algoritmos. Las funciones base más utilizadas aparecen en la Figura 28. Onemax, Plateau y Royal Road tienen un nivel de dificultad creciente. Por su parte, Deceptive es difícil ya que las soluciones más cercanas en bits al óptimo tienen un peor valor de la función objetivo, y esto lleva a LS hacia la solución sin bits en 1 que está muy lejana del óptimo global. Estas funciones base tienen algunos aspectos en común como por ejemplo, que solo recibe el máximo valor de 4 el bloque 1111 con los 4 bits correctos, mientras que para el resto de las cantidades posibles de bits correctos (0, 1, 2, o 3) la función vale 0, 1, 2 o 3. Como hay cuatro valores posibles para cada una de las 4 cantidades posibles de bits correctos, existen 256=$4^4$ funciones base con estas características. Con estas características se generaron 24 funciones aleatoriamente dentro de las 256 posibles.

Estas 24 funciones, junto con las 4 de la Figura 28 conforman el marco experimental de 28 funciones de la Tabla 4. Cada función Fwxyz es la función base que asigna el valor “w” cuando hay 0 bits correctos, “x” cuando hay 1 , “y” cuando hay 2, y “z” cuando hay 3 bits correctos. Las funciones F0123, F0003, F0000 y F3210 se corresponden, respectivamente, con las funciones Onemax, Plateau, Royalroad y Deceptive. En el anexo 7 se detallan las funciones y se resume los grados de dificultad. Las funciones escogidas tienen una variedad representativa, con distintos grados de dificultad.

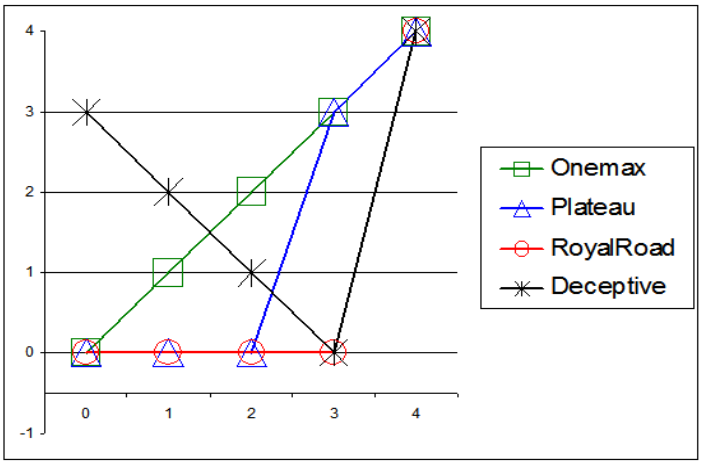



Figura 27: Algunas funciones base.

Tabla 4: Funciones base usadas.

| Funciones de bloque | | | | | | |
|---|---|---|---|---|---|---|
| F0000 | F0111 | F0131 | F1111 | F2000 | F2220 | F3010 |
| F0001 | F0120 | F1001 | F1112 | F2001 | F3000 | F3012 |
| F0003 | F0122 | F1002 | F1120 | F2012 | F3001 | F3102 |
| F0022 | F0123 | F1012 | F1221 | F2200 | F3002 | F3210 |

**Metaheurísticas a comparar**

En los experimentos se comparan las cuatro S-Metaheurísticas Proactivas propuestas con las cuatro S-Metaheurísticas no proactivas originales [162] (RRT, TA, GDA y LS), y tres P-Metaheurísticas (algoritmos evolutivos). En todas se construye una solución inicial asignando a cada uno de los 100 bits de la cadena un valor binario aleatorio de 0 o 1. Como operador de mutación se utiliza la

mutación de un bit, que genera una nueva solución idéntica a la de referencia excepto en un bit, que se escoge aleatoriamente entre los 100 posibles. En el Algoritmo Genético (AG) y las Estrategias Evolutivas (EE) se usó el reemplazo generacional [162], seleccionando la nueva generación entre los M mejores de las P nuevas soluciones (hijos) y las M anteriores (padres). En los AG se usa el cruzamiento uniforme [162]. Los otros parámetros de cada una de las metaheurísticas y el identificador usado para nombrar a cada una aparecen en la Tabla 5. Estos parámetros fueron fijados partiendo de las recomendaciones usuales [18, 162] y ajustados luego, usándose los parámetros que mejores resultados dieron. Para todas las metaheurísticas se usará una ventana de tiempo TW=200. En PLS se utilizó un valor inicial de N=1 (comienza usando la estructura de vecindad de la mutación con cambio de un bit).

Tabla 5: Configuración de metaheurísticas.

| Identificador: Configuración de parámetros |
|---|
| LS: Búsqueda Local o Escalador de Colinas Estocástico, aceptando soluciones iguales o mejores |
| RRT: Record-to-Record-Travel con desviación D=5 |
| GDA: Algoritmo del Gran Diluvio, nivel del agua inicial WL=0, y lluvia R=0.01 |
| TA: Algoritmo de Aceptación por Umbral con umbral T=2 |
| AG-50-100: AG, P=100, M=50, probabilidad de mutación 1 |
| EE-1-5: EE, P= 5 y M=1 |
| EE-20-100: EE, P=100 y M=20 |

**Resultados**

Por cada experimento de las once metaheurísticas probadas en las 28 funciones, fueron realizadas 30 ejecuciones independientes hasta llegar a 10000 evaluaciones. Se registró la mejor solución encontrada en cada una de las ejecuciones. Estos 30 resultados por cada metaheurística en cada función fueron resumidos utilizando la media aritmética. Además, como la prueba de Kruskal-Wallis reveló diferencias entre los algoritmos en cada función, se realizó una comparación entre cada par de algoritmos utilizando la prueba no pareada de Wilcoxon (siempre se trabajó con nivel de significación de 0.05). Cuando una

metaheurísticas es mejor que otra en la prueba de Wilcoxon, la ganadora recibe +1 y la perdedora recibe -1. Debido a esto, si una metaheurística tiene un comportamiento superior a las otras 10 en una función, ella tiene una puntuación perfecta (SW=10). De forma general, este valor SW estará entre -10 y 10, y será utilizada también para resumir el comportamiento del algoritmo en cada función. La pertinencia de pruebas no paramétricas (como la prueba de Wilcoxon y la prueba de Friedman) para la comparación de metaheurísticas, han sido demostradas en [65].

En el anexo 8 se muestran los detalles de los resultados obtenidos por las 11 metaheurísticas en cada una de las 28 funciones. Usando las siguientes métricas, se resumen esos resultados en la Tabla 6 y Tabla 7.

- MA(MA): Media aritmética calculada sobre las 28 medias aritméticas correspondientes a cada una de las funciones (cada una calculada en las 30 ejecuciones).
- ME(MA): Mediana calculada sobre las 28 medias aritméticas correspondientes a cada una de las funciones (cada una calculada en las 30 ejecuciones).
- MA(DT): Media aritmética calculada sobre las 28 desviaciones estándares correspondientes a cada una de las funciones (cada una calculada en las 30 ejecuciones).
- MA(PC): Media aritmética calculada sobre los 28 porcientos de convergencia correspondientes a cada una de las funciones (cada porcentaje indica el porcentaje de las 30 ejecuciones en que se logra alcanzar el óptimo).
- ME(PC): Mediana calculada sobre los 28 porcientos de convergencia correspondientes a cada una de las funciones (cada porcentaje indica el porcentaje de las 30 ejecuciones en que se logra alcanzar el óptimo).
- MA(SW): Media aritmética calculada sobre las 28 sumas de resultados en la prueba de Wilcoxon no pareado (o test de Mann-Whitney) correspondientes a cada una de las funciones (cada prueba considerando las 30 ejecuciones).

- ME(SW): Mediana calculada sobre las 28 sumas de resultados en la prueba de Wilcoxon no pareado (o test de Mann-Whitney) correspondientes a cada una de las funciones (cada prueba considerando las 30 ejecuciones).
- PC: Porcentaje de las 28 funciones en que el algoritmo obtiene convergencia perfecta al óptimo en las 30 ejecuciones.
- SW(MA): Suma de resultados en la prueba de Wilcoxon pareada considerando los 28 valores de media aritmética en cada una de las funciones (cada uno calculado en las 30 ejecuciones de cada función). Se comparan cada par de metaheurísticas y si hay diferencia significativa (0.05) se le suma +1 al que gana, y -1 al que pierde.
- SR Friedman: Suma de los rangos (posición en orden descendente, siendo mejor cuando es menor) obtenidos a partir de la prueba de Friedman, aplicada sobre las 28 medias aritméticas correspondientes a cada una de las funciones (cada una calculada en las 30 ejecuciones).

Tabla 6: Comparación de los resultados en las 5 primeras métricas.

| | MA(MA) | ME(MA) | MA(DT) | MA(PC) | ME(PC) |
|---|---|---|---|---|---|
| AG100 | 90.67 | 91.18 | 3.54 | 12% | 0% |
| EE100 | 90.53 | 94.17 | 2.06 | 28% | 0% |
| EE5 | 88.59 | 92.28 | 2.35 | 31% | 0% |
| GDA | 82.36 | 85.83 | 4.23 | 13% | 0% |
| LS | 83.21 | 86.38 | 2.24 | 32% | 0% |
| PGDA | 92.11 | 98.20 | 1.73 | 50% | 42% |
| PLS | 91.41 | 94.37 | 3.04 | 34% | 3% |
| PRRT | 91.25 | 98.63 | 1.52 | 52% | 55% |
| PTA | 91.60 | 100.00 | 0.95 | 61% | 100% |
| RRT | 78.53 | 79.77 | 3.16 | 0% | 0% |
| TA | 80.83 | 87.68 | 1.32 | 43% | 0% |

Tabla 7: Comparación de los resultados en las 5 últimas métricas.

| | PC | MA(SW) | ME(SW) | SW(MA) | SR Friedman |
|---|---|---|---|---|---|
| AG100 | 0% | -0.25 | 0.5 | 3 | 178.5 |
| EE100 | 14% | 0.93 | 1.5 | 3 | 160 |
| EE5 | 25% | 0.18 | 0 | 0 | 170 |
| GDA | 0% | -3.64 | -4 | -6 | 218.5 |
| LS | 32% | -2.36 | -4 | -7 | 201.5 |
| PGDA | 39% | 3.79 | 4 | 4 | 112 |
| PLS | 18% | 1.79 | 3 | 3 | 154.5 |
| PRRT | 46% | 3.00 | 4.5 | 4 | 115 |
| PTA | 54% | 3.46 | 5 | 3 | 110 |
| RRT | 0% | -5.36 | -7.5 | -7 | 243.5 |
| TA | 43% | -1.54 | -6 | 0 | 184.5 |

En la Tabla 8 se muestra una visión resumida de los resultados de la Tabla 6 y Tabla 7. Para cada una de las 10 métricas anteriores, se ordenaron los algoritmos. En el caso de las métricas MA(MA), ME(MA), MA(PC), ME(PC), MA(SW), ME(SW), PC y SW(MA) los algoritmos se ordenaron descendentemente (mayor es mejor), mientras que para las métricas MA(DT) y SR Friedman se ordenaron ascendentemente (menor es mejor). A partir de estos rangos lo ideal es que las metaheurísticas obtengan los rangos menores.

Tabla 8: Resumen del ordenamiento de los resultados.

| | 1 | 2 | 3 | MA(R) | ME(R) | SW-M |
|---|---|---|---|---|---|---|
| AG100 | 0 | 0 | 1 | 6.9 | 7 | -4 |
| EE100 | 0 | 0 | 1 | 5.5 | 5 | -1 |
| EE5 | 0 | 0 | 0 | 6.4 | 6.5 | -1 |
| GDA | 0 | 0 | 0 | 9 | 9 | -6 |
| LS | 0 | 0 | 0 | 7.5 | 8 | -2 |
| PGDA | 3 | 1 | 4 | 2.5 | 3 | 7 |
| PLS | 0 | 0 | 2 | 4.6 | 4 | 0 |
| PRRT | 1 | 5 | 3 | 2.4 | 2 | 8 |
| PTA | 7 | 2 | 1 | 1.4 | 1 | 8 |
| RRT | 0 | 0 | 0 | 9.9 | 11 | -8 |
| TA | 0 | 1 | 1 | 6.5 | 7.5 | -1 |

La Tabla 8 muestra la cantidad de veces que una metaheurística obtiene una de las tres primeras posiciones de una métrica (rango de 1, 2 y 3). También se muestra la media aritmética y la mediana de los rangos en las columnas MA(R) y ME(R). Con estos 10 rangos para cada metaheurística se realizó la prueba de Friedman reflejando una diferencia significativa (p-value= $2.9*10^{-12}$). A partir de estos resultados, se calculó la suma de resultados en la prueba de Wilcoxon pareada considerando los 10 rangos. Se comparan cada par de metaheurísticas y si hay diferencia significativa (0.05) se le suma +1 al que gana, y -1 al que pierde. Los resultados de la suma se muestran en la columna SW-M.

La Tabla 9 muestra la comparación de las medias aritméticas de las variantes proactivas y no proactivas. Como puede verse, en 16 de las 28 funciones (57,14 %) las cuatro variantes proactivas se comportaron mejor o igual que cada uno de sus pares no proactivas.

Como puede apreciarse, en general las variantes proactivas logran mejores resultados. Solo en la función f1221 ninguna de las variantes proactivas es mejor. Por otra parte, en f0120, f1120, f2220 y f2200 sólo la variante PLS supera a LS, pero las demás variantes proactivas pierden. Hay aspectos comunes en las funciones como que el valor asignado por la función para un valor de tres bits correctos es muy bajo. Debe recordarse que a la luz de NFL no puede aspirarse a obtener una metaheurística superior a las demás. Sin embargo, sí es importante tratar de describir cuándo cada metaheurística funciona bien o mal, aunque el estudio a fondo de por qué esto pasa escapa de los objetivos y del espacio disponible en esta tesis.

Tabla 9: Comparación de las medias aritméticas de las variantes proactivas y no proactivas.

| | Diferencias | | | | Versiones proactivas | |
|---|---|---|---|---|---|---|
| MA | PGDA-GDA | PRRT-RRT | PTA-TA | PLS-LS | Mejor | Mejor o igual |
| f1002 | 37.00 | 34.57 | 34.13 | 23.63 | 4 | 4 |
| f1012 | 27.97 | 27.73 | 30.83 | 16.90 | 4 | 4 |
| f2001 | 45.93 | 43.50 | 43.23 | 30.83 | 4 | 4 |
| f2012 | 0.73 | 18.70 | 47.60 | 1.60 | 4 | 4 |
| f3000 | 7.23 | 10.20 | 12.10 | 3.47 | 4 | 4 |
| f3001 | 1.97 | 8.77 | 9.70 | 1.13 | 4 | 4 |
| f3002 | 23.97 | 21.17 | 21.93 | 15.30 | 4 | 4 |
| f3010 | 41.47 | 40.40 | 42.73 | 36.57 | 4 | 4 |
| f3012 | 5.37 | 13.10 | 23.63 | 5.07 | 4 | 4 |
| f0122 | 5.00 | 23.93 | 41.03 | -0.07 | 3 | 3 |
| f1001 | 2.33 | 13.57 | 0.00 | 10.80 | 3 | 4 |
| f1112 | 1.80 | 24.30 | 47.73 | 0.00 | 3 | 4 |
| f2000 | 12.07 | 23.00 | 0.00 | 8.53 | 3 | 4 |
| f0123 | 0.33 | 20.13 | 33.57 | 0.00 | 3 | 4 |
| f0022 | 4.80 | 20.33 | 43.73 | 0.00 | 3 | 4 |
| f0003 | 0.40 | 13.27 | 12.10 | 0.00 | 3 | 4 |
| f0131 | 10.47 | 10.73 | 0.00 | 7.93 | 3 | 4 |
| f3210 | 1.97 | 0.27 | -1.43 | 4.43 | 3 | 3 |
| f3102 | 0.10 | 11.43 | 42.43 | -0.50 | 3 | 3 |
| f0001 | 1.57 | 12.50 | 0.00 | 0.00 | 2 | 4 |
| f0111 | 11.97 | 19.80 | 0.00 | -0.30 | 2 | 3 |
| f1111 | 14.40 | 16.00 | 0.00 | -0.80 | 2 | 3 |
| f0000 | 20.13 | 22.67 | 0.00 | -1.07 | 2 | 3 |
| f0120 | -1.13 | -19.47 | -43.93 | 20.10 | 1 | 1 |
| f1120 | -0.23 | -17.73 | -43.60 | 21.53 | 1 | 1 |
| f2220 | -1.27 | -21.40 | -43.40 | 19.07 | 1 | 1 |
| f2200 | -2.20 | -19.20 | -38.07 | 10.80 | 1 | 1 |
| f1221 | -1.00 | -16.13 | -14.40 | -5.50 | 0 | 0 |

Otra manera de resumir los resultados puede basarse en la comparación entre variantes proactivas y no proactivas en las métricas indicando cuando la versión proactiva es mejor (P), o cuando no lo es (NP). Esto se presenta en la Tabla 10.

En la Tabla 10 la columna p-value indica la significación en la prueba de los signos [39].

Tabla 10: Comparación entre variantes proactivas y no proactivas en las métricas.

| Comparación | Métricas | | | | | | | | | | Cantidad de veces | | p-value |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PC | MA(MA) | ME(MA) | MA(DT) | MA(SW) | ME(SW) | SW | MA(PC) | ME(PC) | SR Friedman | P | NP | |
| PTA vs TA | P | P | P | P | P | P | P | P | P | P | 10 | 0 | 0.001 |
| PRRT vs RRT | P | P | P | P | P | P | P | P | P | P | 10 | 0 | 0.001 |
| PGDA vs GDA | P | P | P | P | P | P | P | P | P | P | 10 | 0 | 0.001 |
| PLS vs LS | NP | P | P | NP | P | P | P | P | P | P | 8 | 2 | 0.0547 |
| Total | | | | | | | | | | | 38 | 2 | 3.3*10-15 |

De acuerdo con todos los resultados mostrados, los mejores resultados son obtenidos por los algoritmos PGDA, PRRT y PTA, seguido luego por PLS y las dos EE. Usando la prueba de Friedman entre estos algoritmos, aún las diferencias fueron significativas. Sólo dentro del grupo de las tres primeras (PGDA, PRRT y PTA) no se obtuvieron diferencias significativas (p-value=0.0655462) usando significación de 0.05.

Finalmente, la valoración en general de la proactividad se puede analizar comparando las variantes proactivas contra las no proactivas, es decir PTA vs TA, PGDA vs GDA, PRRT vs RRT y PLS vs LS. De acuerdo a las métricas mostradas en la Tabla 6, Tabla 7 y Tabla 10, se puede ver que los resultados de las variantes proactivas son mejores en 38 de las 40 comparaciones, pues solo en dos métricas

PLS no supera a LS. Estos resultados son significativos de acuerdo a las pruebas de los signos (p-value= $3.3*10^{-15}$).

La Tabla 11 muestra un resumen del análisis detallado de los resultados mostrados en la Tabla 9, se presenta la cantidad de funciones en que la variante proactiva es mejor o igual a la no proactiva. Las variantes proactivas, en el promedio obtenido para cada función, son mejores en la mayor parte de las funciones estudiadas que las variantes no proactivas.

Tabla 11: Valoración general de las S-Metaheurísticas proactivas.

| Variante proactiva | PGDA-GDA | PRRT-RRT | PTA-TA | PLS-LS |
|---|---|---|---|---|
| Mejor | 23 (82%) | 23 (82%) | 15 (54%) | 17 (61%) |
| Mejor o igual | 23 (82%) | 23 (82%) | 22 (79%) | 22 (79%) |

En el anexo 8 puede también observarse que en las funciones originales (Onemax, Deceptive, Royal Road y Plateau) al menos 3 de las variantes proactivas superan a las no proactivas.

### 3.4.5 Discusión

Como se puede observar en el modelo de los requisitos tempranos con i* del funcionamiento de una metaheurística se dan las condiciones de dependencia entre dos actores que denotan intencionalidad. En este caso el Optimizador desea optimizar un problema utilizando metaheurísticas sin que estas se estanquen en un óptimo local.

Las metaheurísticas necesitan que el Optimizador ajuste los parámetros y los operadores de vecindad para encontrar la mejor solución. Al darse estas dependencias se puede delegar en las metaheurísticas la meta de no estancarse, para lo cual necesitan un plan para ajustar los parámetros y ajustar las vecindades. De esta forma las metaheurísticas buscarán la mejor solución y evitarán estancarse en un óptimo local, sin que el Optimizador ajuste los parámetros, sólo debe delimitar los datos iniciales. Se puede entonces concluir que la utilización del lenguaje i* y patrones de requisitos para detectar proactividad utilizando los requisitos en i* permitió proponer cuatro metaheurísticas proactivas

que se comportaron mejor que sus variantes no proactivas en la mayoría de las funciones estudiadas.

Resumiendo, los resultados experimentales muestran que la proactividad en el contexto de las S-Metaheurísticas constituye un campo prometedor que debe seguirse desarrollando a partir de los resultados de esta tesis. Algunos de estos resultados fueron presentados en [123].

## 3.5 Conclusiones Parciales

Las principales conclusiones que se pueden extraer de este capítulo son:

- En ambos casos se hizo la pregunta de estudio: ¿Cómo los patrones de requisitos para detectar proactividad utilizando los requisitos en i* permiten identificar requisitos proactivos?, y se pudo constatar la proposición de que la utilización del lenguaje i* y los patrones de requisitos para detectar proactividad permiten identificar requisitos proactivos.
- Al modelar los requisitos tempranos de un observatorio tecnológico con i* y siguiendo la solución propuesta por el patrón de requisitos *Hardgoal why dependency* para detectar proactividad, se pudo detectar requisitos proactivos y delegarlos al sistema que se iba a construir.
- Al modelar los requisitos tempranos de una S-Metaheurística con i* y siguiendo la solución propuesta por los patrones de requisitos *Hardgoal why dependency* y *Resource why dependency* para detectar proactividad se pudo detectar requisitos proactivos y delegarlos en cuatro metaheurísticas, obteniendo una variante proactiva de cada una.
- Se logró construir un sistema multi-agente, que da respuesta a un observatorio tecnológico. La proactividad se logró mediante la observación periódica con el empleo de los patrones de implementación propuestos. Se constata así la proposición de que los patrones de implementación propuestos ayudan a simplificar el desarrollo de un observatorio tecnológico proactivo y de esta forma dar respuesta a la pregunta de estudio: ¿Cómo los patrones de

implementación propuestos simplifican el desarrollo de un sistema informático proactivo?

- Los patrones de requisitos para detectar proactividad permitieron detectar requisitos proactivos en dos unidades de análisis con diferencias marcadas.
- Las cuatro metaheurísticas proactivas desarrolladas se comportaron en promedio mejor que sus variantes no proactivas en la mayoría de las funciones estudiadas.

# *Conclusiones y Recomendaciones*

# *Conclusiones*

Las conclusiones a las que se pueden arribar de los resultados de la tesis son:

- Del estudio bibliográfico se puedo constatar que:
  - ✓ La proactividad es una característica deseable en algunos sistemas informáticos y las metodologías orientadas a objetos y orientadas a agentes no presentan pasos para identificar requisitos proactivos.
  - ✓ Para detectar la proactividad se necesita conocer las causas de las dependencias (los por qué) y la intencionalidad de las relaciones.
  - ✓ Los patrones en la orientación a objetos y a agentes no hacen énfasis en la proactividad.
- Como resultado de la investigación se concluye que:
  - ✓ Los patrones de requisitos formulados, basados en los modelos de i*, permiten detectar requisitos proactivos.
  - ✓ Los patrones de implementación formulados permiten construir sistemas informáticos proactivos, apoyándose la observación periódica y en la tecnología de agentes JADE.
  - ✓ El estudio de casos permitió evaluar cómo los patrones de requisitos propuestos para detectar proactividad permiten identificar requisitos proactivos y cómo los patrones de implementación propuestos simplifican el desarrollo de un sistema informático proactivo.
  - ✓ Utilizando los patrones de requisitos y de implementación formulados se construyó un sistema multi-agente, que da respuesta a un observatorio tecnológico proactivo.
  - ✓ Utilizando los patrones de requisitos formulados se diseñaron cuatro S-Metaheurísticas proactivas que se comportaron, en general, mejor que sus variantes no proactivas y otras P-Metaheurísticas, en los experimentos realizados.

# *Recomendaciones*

- Desarrollar otros patrones de implementación que permitan simplificar el trabajo con las cadenas de mando en un sistema multi-agente, para lograr la inmediatez de una acción dado un cambio.
- Profundizar en la ventajas de la simulación basada en agentes a partir de los requisitos en i*.
- Extender el uso de la proactividad para metaheurísticas poblacionales.
- Estudiar qué características de los problemas hacen más o menos adecuado el empleo de las metaheurísticas en ellos.

# *Referencias Bibliográficas*


1. *Grundlagen des V-Modells, V-Modell® XT V 1.3*, Der Beauftragte für Informations technik, Reporte Técnico, 2008. Disponible en: http://ftp.tu-clausthal.de/ftp/institute/informatik/v-modell-xt/Releases/1.3/V-Modell-XT-Gesamt.pdf.
2. Aalst, Wil M.P. van der, *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. 1st ed., Springer Berlin Heidelberg, Berlín, 2011.
3. Adolph, Steve; Paul Bramble; Alistair Cockburn, *Patterns for Effective Use Cases*. illustrated ed., Addison-Wesley, Boston, 2003.
4. Alechina, Natasha; Mehdi Dastani; Brian Logan; John-Jules Ch. Meyer, *Reasoning about plan revision in BDI agent programs*, Theoretical Computer Science. 412(44): 6115-6134, 2011.
5. Alexander, Christopher; Sara Ishikawa; Murray Silverstein, *A Pattern Language: Towns, Buildings, Construction*. 21th ed., Oxford University Press, New York, 1977.
6. Ali, Raian; Fabiano Dalpiaz; Paolo Giorgini, *A goal-based framework for contextual requirements modeling and analysis*, Requirements Engineering. 15(4): 439-458, 2010.
7. Ammann, P.; J. Offutt, *Introduction to Software Testing*. 1st ed., Cambridge University Press, New York, 2008.
8. An, Yuan; Prudence W. Dalrymple; Michelle Rogers; Patricia Gerrity; Jennifer Horkoff; Eric Yu, *Collaborative social modeling for designing a patient wellness tracking system in a nurse-managed health care center*, en: 4th International Conference on Design Science Research in Information Systems and Technology, Philadelphia: 2009.
9. Aridor, Yariv; Danny B. Lange, *Agent design patterns: elements of agent application design*, en: Second international conference on Autonomous agents, Minnesota: 1998.
10. Badillo, A.R.; J.J. Ruiz; C. Cotta; A.J. Fernández-Leiva, *On user-centric memetic algorithms*, Soft Computing 17(2): 285-300, 2013.
11. Banks, Jerry; John S. Carson II; Barry L. Nelson; David M. Nicol, *Discrete-Event System Simulation*. 5th ed., Prentice Hall, 2009.
12. Bauer, Bernhard; JörgP Müller; James Odell, *Agent UML: A Formalism for Specifying Multiagent Software Systems*, en Agent-Oriented Software Engineering, Lecture Notes in Computer Science, Springer Berlin Heidelberg. 1957: 91-103, 2001.
13. Beck, Kent, *Implementation patterns*. 1st ed., Addison-Wesley, Boston, 2008.
14. Bellifemine, Fabio; Federico Bergenti; Giovanni Caire; Agostino Poggi, *JADE-A Java Agent Development Framework*, en Multi-Agent Programming Languages, Platforms

and Applications, Multiagent Systems, Artificial Societies, and Simulated Organizations, Springer Science+Business Media. 15: 125-147, 2005.

15. Bellifemine, Fabio Luigi; Giovanni Caire; Dominic Greenwood, *Developing Multi-Agent Systems with JADE*. 1st ed., John Wiley & Sons, Ltd, Chichester, 2007.

16. Bergenti, Federico; Marie-Pierre Gleizes; Franco Zambonelli, *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*. 1st ed., Springer, Berlín, 2004.

17. Bertolini, D.; A. Novikau; A. Susi; A. Perini, *TAOM4E: an Eclipse ready tool for Agent-Oriented Modeling. Issue on the development process*, Fondazione Bruno Kessler-irst, Reporte Técnico, 2006. Disponible en: http://selab.fbk.eu/taom/file/demo/TAOM4E_technical_report.pdf.

18. Birattari, Mauro, *Tuning Metaheuristics: A Machine Learning Perspective*. 1st ed., Springer, Brussels, 2009.

19. Birattari, Mauro; Thomas Stutzle; Luis Paquete; Klaus Varrentrapp, *A Racing Algorithm for Configuring Metaheuristics*, en: Genetic and Evolutionary Computation Conference (GECCO' 2002), New York: 2002.

20. Blum, Christian; Maria José Blesa Aguilera; Andrea Roli; Michael Sampels, *Hybrid Metaheuristics: An Emerging Approach to Optimization*. 1st ed., Springer-Verlag, Berlin, 2008.

21. Blum, Christian; Andrea Roli, *Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison*, ACM Computing Surveys. 35(3): 268–308, 2003.

22. Blum, Christian; Andrea Roli, *Hybrid Metaheuristics: An Introduction*, en Hybrid Metaheuristics, Studies in Computational Intelligence, Springer Berlin Heidelberg. 114: 1-30, 2008.

23. Bresciani, Paolo; Anna Perini; Paolo Giorgini; Fausto Giunchiglia; John Mylopoulos, *Tropos: An Agent-Oriented Software Development Methodology*, Autonomous Agents and Multi-Agent Systems. 8(3): 203-236, 2004.

24. Budd, Timothy, *An introduction to object-oriented programming*. 3rd ed., Addison-Wesley, 2002.

25. Castro, Jaelson; Paolo Giorgini; Stefanie Kethers; John Mylopoulos, *Tropos: A Requirements-Driven Methodology for Agent-Oriented Software*, en Agent-Oriented Methodologies, Idea Group Inc. 2005.

26. Ceballos, Héctor G.; Ramón F. Brena, *Combinando acceso local y global a Ontologías en Sistemas Multiagentes*, Revista Iberoamericana de Sistemas, Cibernética e Informática. 2(1): 13-22, 2005.

27. Cossentino, Massimo; Giancarlo Fortino; Marie Pierre Gleizes; Juan Pavón, *Simulation-based design and evaluation of multi-agent systems*, Simulation Modelling Practice and Theory. 18(10): 1425-1427, 2010.

28. Cossentino, Massimo; Marie-Pierre Gleizes; Ambra Molesini; Andrea Omicini, *Processes Engineering and AOSE*, en Agent-Oriented Software Engineering X, Lecture Notes in Computer Science, Springer Berlin Heidelberg. 6038: 191-212, 2011.

29. Cossentino, Massimo; Luca Sabatucci; Antonio Chella, *Patterns Reuse in the PASSI Methodology*, en Engineering Societies in the Agents World IV, Lecture Notes in Computer Science, Springer Berlin Heidelberg. 3071: 294-310, 2004.

30. Cotta, Carlos; Marc Sevaux; Kenneth Sorensen, *Adaptive and Multilevel Metaheuristics*. 1st ed., Springer-Verlag Berlin Heidelberg, 2008.

31. Crant, J. Michael, *Proactive Behavior in Organizations*, Journal of Management. 26(3): 435-462, 2000.

32. Cruz, Carlos; Juan R. González; David A. Pelta, *Optimization in dynamic environments: a survey on problems, methods and measures*, Soft Computing. 15(7): 1427–1448, 2011.

33. Chella, Antonio; Massimo Cossentino; Luca Sabatucci, *Tools and patterns in designing multi-agent systems with PASSI*, WSEAS Transactions on Communications. 3(1): 352-358, 2004.

34. Chen, Hong-Mei; Rick Kazman; Opal Perry, *From Software Architecture Analysis to Service Engineering: An Empirical Study of Methodology Development for Enterprise SOA Implementation*, IEEE Transactions On Services Computing. 3(2): 145-160, 2010.

35. Dalpiaz, F.; E. Paja; P. Giorgini, *Security Requirements Engineering via Commitments*, en: First Workshop on Socio-Technical Aspects in Security and Trust, STAST'11, Milan: 2011.

36. de la Vega, Iván, *Tipología de Observatorios de Ciencia y Tecnología. Los casos de América Latina y Europa*, Revista Española De Documentación Científica. 30(4): 545-552, 2007.

37. DeLoach, S.; M. Kumar, *Multi-Agent Systems Engineering: An Overview and Case Study*, en Agent-Oriented Methodologies, Idea Group Inc. 2005.

38. DeLoach, ScottA, *Engineering Organization-Based Multiagent Systems*, en Software Engineering for Multi-Agent Systems IV, Lecture Notes in Computer Science, Springer Berlin Heidelberg. 3914: 109-125, 2006.

39. Derrac, Joaquín; Salvador Garcí; Daniel Molina; Francisco Herrera, *A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms*, Swarm and Evolutionary Computation. 1(1): 3-18, 2011.

40. Dignum, Frank; Virginia Dignum; John Thangarajah; Lin Padgham; Michael Winikoff, *Open Agent Systems ???*, en Agent-Oriented Software Engineering VIII, Lecture Notes in Computer Science, Springer Berlin Heidelberg. 4951: 73-87, 2008.

41. Doerner, Karl F.; Michel Gendreau; Peter Greistorfer; Walter J. Gutjahr; Richard F. Hartl; Marc Reimann, *Metaheuristics: Progress in Complex Systems Optimization*. 1st ed., Springer Science+Business Media, 2007.

42. Dubé, Line; Guy Paré, *Rigor in information systems positivist case research: current practices, trends, and recommendations*, MIS Quarterly. 27(4): 597-635, 2003.

43. Elahi, G.; E. Yu; N. Zannone, *Security risk management by qualitative vulnerability analysis*, en: 3rd International Workshop on Security Measurements and Metrics, Metrisec 2011, Canada: 2011.

44. Evitts, Paul, *A UML pattern language*. Macmillan Technical Publishing, 2000.

45. Farcas, Alan, *KAWAX. Observatorio Chileno de Ciencia, Tecnología e Innovación. Estado actual y perspectivas*, en: II Seminario Internacional sobre Indicadores de Ciencia, Tecnología e Innovación Santiago de Chile: 2006.

46. Ferber, Jacques, *Multi-agent systems: an introduction to distributed artificial intelligence*. 1st ed., Addison-Wesley, 1999.

47. Fernández, Felix Oscar, *Un Modelo de Sistema de gestión Documental Colaborativo y Supervisado (SICLOS)*, Tesis Doctoral, Instituto Superior Politécnico José Antonio Echeverría, La Habana, Cuba, 2007.

48. Fernández, Perla B., *Comparación del Análisis de Requisitos en i* y RUP en la Intranet de Informática*, Tesis de Diploma, Instituto Superior Politécnico "José Antonio Echeverría", La Habana, Cuba, 2010.

49. FIPA, The Foundation For Intelligent Phisical Agents. [Accedido: noviembre 2012], http://www.fipa.org.

50. FIPA, *FIPA ACL Message Structure Specification*, FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS, Reporte Técnico, 2003. Disponible en: http://www.fipa.org/specs/fipa00061/SC00061G.html.

51. FIPA, *FIPA Agent Management Specification*, Foundation for Intelligent Physical Agents, Reporte Técnico, 2003. Disponible en: http://www.fipa.org/specs/fipa00023/XC00023H.html.

52. FIPA, *FIPA Communicative Act Library Specification*, FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS, Reporte Técnico, 2003. Disponible en: http://www.fipa.org/specs/fipa00037/SC00037J.html.

53. FIPA, *FIPA Methodology: Glossary of Terms*, FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS, Reporte Técnico 2003. Disponible en: http://www.pa.icar.cnr.it/cossentino/FIPAmeth/docs/glossary/glossary_1_0.pdf.

54. FIPA, *FIPA Subscribe Interaction Protocol Specification*, FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS, Reporte Técnico, 2003. Disponible en: http://www.fipa.org/specs/fipa00035/SC00035H.pdf.

55. Fowler, Martin, *UML distilled*. 3rd ed., Addison-Wesley Professional, 2004.

56. Fox, John, *Understanding intelligent agents: analysis and synthesis*, AI Communications. 16(3): 139, 2003.

57. Frankl, Viktor, *Man's Search for Meaning*. Beacon Press, 2006.

58. Franklin, Stan; Art Graesner, *It is an Agent, or just a Program?: A Taxonomy for Autonomous Agents*, en: Intelligent Agents III, Agent Theories, Architectures, and Languages: 1996.

59. Fuentes-Fernández, Rubén; Jorge J. Gómez-Sanz; Juan Pavón, *Requirements elicitation and analysis of multiagent systems using activity theory*, IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans. 39(2): 282-298, 2009.

60. Fuentes-Fernández, Rubén; JorgeJ Gómez-Sanz; Juan Pavón, *Understanding the human context in requirements elicitation*, Requirements Engineering. 15(3): 267-283, 2010.

61. Fuentes, Rubén, *Teoría de Actividad dara el Desarrollo de Sistemas Multi-Agente*, Tesis Doctoral, Universidad Complutense de Madrid, Madrid, 2004.

62. Gallego, D. ; E. Barra; S. Aguirre; G. Huecas, *A model for generating proactive context-aware recommendations in e-Learning systems*, en: 42nd Annual Frontiers in Education Conference, FIE 2012, Seattle: 2012.

63. Gamma, Erich; Richard Helm; Ralph Johnson; John Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*. Pearson Education, 2004.

64. García, Omarys, *Las Pruebas en los Sistemas Agentes: Una aproximación inicial*, Tesis de Diplomado, Instituto Superior Politécnico "José Antonio Echeverría", 2007.

65. García, Salvador; Daniel Molina; Manuel Lozano; Francisco Herrera, *A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 Special Session on Real Parameter Optimization*, Journal of Heuristics. 15(6): 617-644, 2009.

66. Gascueña, JoséM; Antonio Fernández-Caballero, *Prometheus and INGENIAS Agent Methodologies: A Complementary Approach*, en Agent-Oriented Software Engineering IX, Lecture Notes in Computer Science, Springer Berlin Heidelberg. 5386: 131-144, 2009.

67. Georgeff, M.; A. Rao, *Rational Software Agents: From Theory to Practice*, en Agent TechnologySpringer Berlin Heidelberg. 139-160, 1998.

68. German, Ernesto; Leonid Sheremetov, *An Agent Framework for Processing FIPA-ACL Messages Based on Interaction Models*, en Agent-Oriented Software Engineering VII, Lecture Notes in Computer Science, Springer Berlin Heidelberg. 4951: 88-102, 2008.

69. Ghorbani, J.; M.A. Choudhry; A. Feliachi, *Real-time multi agent system modeling for fault detection in power distribution systems* en: North American Power Symposium, Champaign: 2012.

70. Gil, Joseph; Itay Maman, *Implementation Patterns*, Department of Computer Science Technion-Israel Institute of Technology, Reporte Técnico, 2004. Disponible en: http://www.cs.technion.ac.il/~imaman/stuff/ip-ecoop05.pdf.

71. Gómez-Sanz, Jorge J.; Carlos R. Fernández; Javier Arroyo, *Model driven development and simulations with the INGENIAS agent framework*, Simulation Modelling Practice and Theory. 18(10): 1468–1482, 2010.

72. Gómez-Sanz, Jorge J.; Rubén Fuentes; Juan Pavón; Iván García-Magariño, *INGENIAS development kit: a visual multi-agent system development environment*, en: 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008), Estoril, Portugal: 2008.

73. Gómez-Sanz, JorgeJ; Juan Botía; Emilio Serrano; Juan Pavón, *Testing and Debugging of MAS Interactions with INGENIAS*, en Agent-Oriented Software Engineering IX, Lecture Notes in Computer Science, Springer Berlin Heidelberg. 5386: 199-212, 2009.

74. González, Juan R.; Carlos Cruz; Ignacio G. del Amo; David A. Pelta, *An adaptive multiagent strategy for solving combinatorial dynamic optimization problems*, en: Nature Inspired Cooperative Strategies for Optimization (NICSO 2011), Cluj-Napoca: 2012.

75. Gordijn, Jaap; Michael Petit; Roel Wieringa, *Understanding Business Strategies of Networked Value Constellations Using Goal- and Value Modeling*, en: 14th IEEE International Requirements Engineering Conference, 2006.

76. Grant, A. M.; S. J. Ashford, *The dynamics of proactivity at work*, Research in Organizational Behavior. 28(-): 3-34, 2008.

77. Grau, Gemma; Xavier Franch, *On the Adequacy of i* Models for Representing and Analyzing Software Architectures*, en Advances in Conceptual Modeling – Foundations and Applications, Lecture Notes in Computer Science, Springer Berlin Heidelberg. 4802: 296-305, 2007.

78. Günther, Christian W.; Eric Verbeek, *XES Standard Definition*, Technische Universiteit Eindhoven University of Technology, Reporte Técnico, 2012. Disponible en: http://www.xes-standard.org/_media/xes/xesstandarddefinition-1.4.pdf.

79. Hadar, Irit; Tsvi Kuflik; Anna Perini; Iris Reinhartz-Berger; Filippo Ricca; Angelo Susi, *An Empirical Study of Requirements Model Understanding: Use Case vs. Tropos Models*, en: 2010 ACM Symposium on Applied Computing, SAC'10, Sierre: 2010.

80. Hadfeg, Yahima, *Transformación de artefactos i* a UML 2.0: Propuesta Inicial*, Tesis de Diplomado, Instituto Superior Politécnico "José Antonio Echeverría", 2009.

81. Hadfeg, Yahima, *Ingenias: Incorporando Análisis de Requisitos*, Tesis de Maestría, Instituto Superior Politécnico "José Antonio Echeverría", La Habana, Cuba, 2011.

82. Henderson-Sellers, B.; J. Debenham; Q. N. N. Tran; M. Cossentino; G. Low, *Identification of Reusable Method Fragments from the PASSI Agent-Oriented*

*Methodology*, en Agent-Oriented Information Systems III, Lecture Notes in Computer Science, Springer Berlin Heidelberg. 3529: 95-110, 2006.

83. Henderson-Sellers, Brian, *From Object-Oriented to Agent-Oriented Software Engineering Methodologies*, en Software Engineering for Multi-Agent Systems III, Lecture Notes in Computer Science, Springer Berlin Heidelberg. 3390: 1-18, 2005.
84. Henderson-Sellers, Brian; Paolo Giorgini, *Agent-Oriented Methodologies*. 1st ed., Idea Group Inc, Hershey, 2005.
85. Hevner, Alan; Samir Chatterjee, *Design Research in Information Systems, Theory and Practice*. 1st ed., Springer, 2010.
86. Hevner, Alan R.; Sudha Ram; Salvatore T. March; Jinsoo Park, *Design Science in Information Systems Research*, MIS Quarterly. 28(1): 75-105, 2004.
87. Hidalgo, A. G.; J. J. Gomez-Sanz; J. P. Mestras, *Workflow Modelling with INGENIAS methodology*, en: 5th IEEE International Conference on Industrial Informatics 2007.
88. Hofstede, Arthur H. M. ter; Wil M. P. van der Aalst; Michael Adams; Nick Russell, *Modern Business Process Automation: YAWL and its Support Environment*. 1st ed., Springer Heidelberg, 2010.
89. Horkoff, Jennifer; Eric Yu, *Comparison and evaluation of goal-oriented satisfaction analysis techniques*, Requirements Engineering. 18(3): 199-222, 2013.
90. Jacobson, I.; G. Booch; J. Rumbaugh, *The Unified Software Development Process*. reprint ed., Prentice Hall, 2012.
91. Jennings, Nicholas R., *On agent-based software engineering*, Artificial Intelligence. 117(2): 277-296, 2000.
92. Jennings, Nicholas R., *An agent-based approach for building complex software systems*, Communications of the ACM. 44(4): 35-41, 2001.
93. Jennings, Nicholas R.; Michael Wooldridge, *Agent-Oriented Software Engineering*, en: 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World: Multi-Agent System Engineering (MAAMAW-99). pp. Valencia, 1999.
94. Jones, Terry; Stephanie Forrest, *Fitness distance correlation as a measure of problem difficulty for genetic algorithms*, en: Sixth International Conference on Genetic Algorithms, San Francisco: 1995.
95. Julian, V.; V. Botti, *Agentes Inteligentes: el siguiente paso en la Inteligencia Artificial*, Novática. 145(-): 95-99, 2000.
96. Khaitan, Siddhartha Kumar; James D. McCalley; Arun Somani, *Proactive task scheduling and stealing in master-slave based load balancing for parallel contingency analysis*, Electric Power Systems Research. 103(-): 9-15, 2013.
97. Klau, G.W.; N. Lesh; J. Marks; M. Mitzenmacher, *Human-guided search*, Journal of Heuristics. 16(3): 289-310, 2010.

98. Lang, H.; W. Minker, *A collaborative Web-based help-system* en: 2nd International Conference on Web Intelligence, Mining and Semantics, WIMS 2012, Craiova: 2012.

99. Lang, Helmut; Melina Klepsch; Florian Nothdurft; Tina Seufert; Wolfgang Minker, *The Influence of Proactivity on Interactive Help Agents*, en Human Factors in Computing and Informatics, Lecture Notes in Computer Science, Springer Berlin Heidelberg. 7946: 748-767, 2013.

100. Law, Averill M.; W. David Kelton, *Simulation Modeling and Analysis*. 3rd ed., McGraw Hill, 2000.

101. Lee, D.; P. Brusilovsky, *Proactive: Comprehensive access to job information*, Journal of Information Processing Systems. 8(4): 721-738, 2012.

102. Lepagnot, Julien; Amir Nakib; Hamouche Oulhadj; Patrick Siarry, *A New multiagent Algorithm for Dynamic Continuous optimization*, International Journal of Applied Metaheuristic Computing. 1(1): 16-38, 2010.

103. Lessard, Lysanne; Eric Yu, *Using Design Science Research to Develop a Modeling Technique for Service Design*, en Design Science Research in Information Systems. Advances in Theory and Practice, Lecture Notes in Computer Science, Springer Berlin Heidelberg. 7286: 66-77, 2012.

104. Li, Jiao; Yuqiang Feng, *Construction of Multi-Agent System for Decision Support of Online Shopping*, en Emerging Research in Artificial Intelligence and Computational Intelligence, Communications in Computer and Information Science, Springer Berlin Heidelberg. 237: 318-324, 2011.

105. Li, Minyi; Quoc Bao Vo; Ryszard Kowalczyk; Sascha Ossowski; Gregory Kersten, *Automated negotiation in open and distributed environments*, Expert Systems with Applications. 40(15): 6195-6212, 2013.

106. Lindgren, Helena; Dipak Surie; Ingeborg Nilsson, *Agent-Supported Assessment for Adaptive and Personalized Ambient Assisted Living*, en Trends in Practical Applications of Agents and Multiagent Systems, Advances in Intelligent and Soft Computing, Springer Berlin Heidelberg. 90: 25-32, 2011.

107. Lopes, Fernando; Michael Wooldridge; Augusto Q. Novais, *Negotiation among autonomous computational agents: principles, analysis and challenges*, Artificial Intelligence Review. 29(1): 1-44, 2008.

108. López-Nores, M.; J. García-Duque; J.J. Pazos-Arias; Y. Blanco-Fernández; M. Ramos-Cabrer; A. Gil-Solla; R.P. Díaz-Redondo; A. Fernández-Vilas, *KEPPAN: Knowledge exploitation for proactively-planned ad-hoc networks*, Journal of Network and Computer Applications. 32(6): 1194-1209, 2009.

109. López, Carlos R., *Metodología para la Sistematización de los Servicios de Consultoría TI: Aplicación al Sector de la Manufactura*, Tesis Doctoral, Universidad de Alicante, 2011.

110. McNaull, J.; J.C. Augusto; M. Mulvenna; P. McCullagh, *Multi-agent System Feedback and Support for Ambient Assisted Living*, en: 8th International Conference on Intelligent Environments, Guanajuato 2012.

111. Min Yuh, Day; Lu Chun Hung; David Yang Jin Tan; Chiou Guey Fa; Ong Chorng Shyong, *Designing an Ontology-Based Intelligent Tutoring Agent with Instant Messaging*, en: Fifth IEEE International Conference on Advanced Learning Technologies, ICALT´05 Kaohsiung: 2005.

112. Miranda, Dainiel; Taysir B. Taha, *Propuesta de transformación de los requisitos tempranos y tardíos de i* al modelo del negocio de RUP*, Diploma, Instituto Superior Politécnico "José Antonio Echeverría", La Habana, Cuba, 2013.

113. Molesini, Ambra; Enrico Denti; Andrea Omicini, *Agent-based conference management: a case study in SODA*, International Journal of Agent-Oriented Software Engineering. 4(1): 1-31, 2010.

114. Moraïtis, Pavlos; Elefthreria Petraki; Nikolaos I. Spanoudakis, *Engineering JADE Agent with the Gaia Methodology*, en Agent Technologies, Infrastructures, Tools, and Applications for e-Services, Lecture Notes in Computer Science, Springer-Verlag. 2592: 77-91, 2003.

115. Morandini, Mirko; DuyCu Nguyen; Anna Perini; Alberto Siena; Angelo Susi, *Tool-Supported Development with Tropos: The Conference Management System Case Study*, en Agent-Oriented Software Engineering VIII, Lecture Notes in Computer Science, Springer Berlin Heidelberg. 4951: 182-196, 2008.

116. Morcillo, P., *Vigilancia e inteligencia competitiva: fundamentos e implicaciones*, Revista de Investigación en Gestión de la Innovación y Tecnología. Vigilancia Tecnológica(17): 2003.

117. Moreno, M.; Z. Fernández; M. Lorenzo; A. Rosete; M. Delgado; Y. Hadfeg, *Integración de Metodologías Orientadas a Agentes*, Revista Cubana de Ciencias Informáticas. 1(4): 4-19, 2007.

118. Moreno, M.; A. Rosete; A. Simón; R. Valdés; E. Leyva; R. SocorroJ. Pina; A. García, *Ingeniería de Software Orientada a Agentes: los roles y las metodologías*, Revista de Ingeniería Industrial. 2006(2): 21-29, 2006.

119. Moreno, Mailyn, *Metodologías Orientadas a Agentes: un estudio comparativo*, Tesis de Maestría, Instituto Superior Politécnico "José Antonio Echeverría", Ciudad de la Habana, 2006.

120. Moreno, Mailyn; Alternán Carrasco; Alejandro Rosete; Martha D. Delgado, *Apoyo a la toma de decisiones en un Observatorio Tecnológico incorporando proactividad*, Revista de Ingeniería Industrial. 34(3): 2013.

121. Moreno, Mailyn; Alternán Carrasco; Alejandro Rosete; Martha D. Delgado, *Patrones de Implementación para Incluir Comportamientos Proactivos*, Polibits. January-June 2013(47): 73-87, 2013.

122. Moreno, Mailyn; Juan Pavón; Alejandro Rosete, *Testing in Agent Oriented Methodologies*, Lecture Notes in Computer Science. 5518(-): 138-145, 2009.

123. Moreno, Mailyn; Alejandro Rosete; Juan Pavón, *An agent based implementation of proactive S-Metaheuristics*, Lecture Notes in Computer Science. 8073(-): 1-10, 2013.

124. Myers, G. J., *The Art of Software Testing*. 2nd ed., John Wiley & Sons, New Jersey, 2004.

125. Mylopoulos, John; Alex Borgida; Matthias Jarke; Manolis Koubarakis, *Telos: representing knowledge about information systems*, ACM Transactions on Information Systems. 8(4): 325-362, 1990.

126. Nguyen, Cu Duy, *Testing Techniques for Software Agents*, Tesis Doctoral, University of Trento, 2008.

127. Nguyen, Cu Duy; Anna Perini; Paolo Tonella, *Goal-oriented testing for MASs*, International Journal of Agent-Oriented Software Engineering. 4(1): 79-109, 2010.

128. Nguyen, Duy C.; Anna Perini; Paolo Tonella, *A Goal-Oriented Software Testing Methodology*, en Agent-Oriented Software Engineering VIII, Lecture Notes in Computer Science, Springer Berlin Heidelberg. 4951: 58-72, 2008.

129. O'Malley, Scott A.; Scott A. DeLoach, *Determining When to Use an Agent-Oriented Software Engineering Paradigm*, en Agent-Oriented Software Engineering II, Lecture Notes in Computer Science, Springer Berlin Heidelberg. 2222: 188-205, 2002.

130. Offermann, Philipp; Sören Blom; Marten Schönherr; Udo Bub, *Artifact Types in Information Systems Design Science – A Literature Review*, en Global Perspectives on Design Science Research, Lecture Notes in Computer Science, Springer Berlin Heidelberg. 6105: 77-92, 2010.

131. Oluyomi, Ayodele; Shanika Karunasekera; Leon Sterling, *An Agent Design Pattern Classification Scheme: Capturing the Notions of Agency in Agent Design Patterns*, en: 1th Asia-Pacific Software Engineering Conference, 2004.

132. Padgham, Lin; John Thangarajah; Michael Winikoff, *The Prometheus Design Tool – A Conference Management System Case Study*, en Agent-Oriented Software Engineering VIII, Lecture Notes in Computer Science, Springer Berlin Heidelberg. 4951: 197-211, 2008.

133. Parker, Sharon K.; Uta K. Bindl; Karoline Strauss, *Making Things Happen: A Model of Proactive Motivation*, Journal of Management. 36(4): 827-856, 2010.

134. Parker, Sharon K.; Helen M. Williams; Nick Turner, *Modeling the Antecedents of Proactive Behavior at Work*, Journal of Applied Psychology. 91(3): 636-652, 2006.

135. Parunak, H. Van Dyke, *Agent-Based and Individual-Based Modeling: A Practical Introduction*, Journal of Artificial Societies and Social Simulation. 15(3): 2012.

136. Pavón, Juan; Jorge Gómez-Sanz; Rubén Fuentes, *The INGENIAS Methodology and Tools*, en Agent-Oriented Methodologies, Idea Group Inc. 2005.

137. Pavon, Juan; Candelaria Sansores; Jorge J. Gomez-Sanz, *Modelling and simulation of social systems with INGENIAS*, International Journal of Agent-Oriented Software Engineering. 2(2): 196-221, 2008.

138. Pelta, David; Carlos Cruz; Juan R. González, *A study on diversity and cooperation in a multiagent strategy for dynamic optimization problems*, International Journal of Intelligent Systems. 24(4): 844-861, 2009.

139. Peña Garcia-Franco, Rafael de la, *Estudio de mercado. El sector de componentes de automoción en Rumanía*, Oficina Económica y Comercial de España en Bucarest, Reporte Técnico, 2012. Disponible en: http://www.observatorioplastico.com/detalle_publicacion.php?id_publicacion=229.

140. Pressman, Roger S., *Software engineering: a practitioner's approach*. 7th ed., McGraw-Hill Higher Education, 2010.

141. Rey Vázquez, Lara, *Informe APEI sobre vigilancia tecnológica*, Asociación Profesional de Especialistas en Información, Reporte Técnico, 2009. Disponible en: http://eprints.rclis.org/17578.

142. Ruey Shun, Chen; Chen Duen Kai, *Apply ontology and agent technology to construct virtual observatory*, Expert Systems with Applications. 34(3): 2019–2028, 2008.

143. Rumbaugh, James; Ivar Jacobson; Grady Booch, *The Unified Modeling Language Reference Manual*. 2nd reprint ed., Addison-Wesley, 2010.

144. Russell, Stuart; Peter Norvig, *Artificial Intelligence: A Modern Approach*. 3rd, illustrated ed., Prentice Hall, 2010.

145. Sabatucci, Luca; Massimo Cossentino; Salvatore Gaglio, *A Semantic Description For Agent Design Patterns*, en: Sixth International Workshop "From Agent Theory to Agent Implementation" (AT2AI-6) at The Seventh International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2008), Estoril: 2008.

146. Sauvage, Sylvain, *Conception de systèmes multi-agents: un thésaurus de motifs orientés agent*, Tesis Doctoral, Université de Caen/Basse-Normandie, 2003.

147. Sauvage, Sylvain, *Agent Oriented Design Patterns: A Case Study*, en: Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 3, New York, USA: 2004.

148. Sauvage, Sylvain, *Design Patterns for Multiagent Systems Design*, en MICAI 2004: Advances in Artificial Intelligence, Lecture Notes in Computer Science, Springer Berlin Heidelberg. 2972: 352-361, 2004.

149. Shalloway, Alan; James J. Trott, *Design Patterns Explained: A New Perspective on Object-Oriented Design*. 2nd ed., Addison-Wesley, 2004.

150. Shoham, Yoav, *Agent-oriented programming*, Artificial Intelligence. 60(-): 51-92, 1993.

151. Shoham, Yoav; Kevin Leyton-Brown, *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. 1st ed., Cambridge University Press, 2008.

152. Siarry, Patrick; Zbigniew Michalewicz, *Advances in Metaheuristics for Hard Optimization*. 1st ed., Springer-Verlag, Berlin Heidelberg, 2008.

153. Sleiman, Kinane, *Comparación del Análisis de Requisitos en i* y RUP del Módulo Control de Asistencias y Notas Parciales*, Diploma, Instituto Superior Politécnico "José Antonio Echeverría", La Habana, Cuba, 2010.

154. Sommerville, Ian, *Integrated Requirements Engineering: A Tutorial*, IEEE Software. 22(1): 16-23, 2005.

155. Sommerville, Ian, *Construction by Configuration: Challenges for Software Engineering Research and Practice*, en: 19th Australian Conference on Software Engineering (ASWEC 2008), 2008.

156. Sommerville, Ian, *Software Engineering*. 9th ed., Addison-Wesley, 2010.

157. Sommerville, Ian; Pete Sawyer, *Requirements Engineering: A Good Practice Guide*. 1st ed., John Wiley & Sons, 1997.

158. Spanoudakis, Nikolaos I.; Pavlos Moraitis, *Modular JADE Agents Design and Implementation Using ASEME* en: 2010 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IAT 2010, Toronto: 2010.

159. Srinivasan, S.; Jagjit Singh; Vivek Kumar, *Multi-agent based decision Support System using Data Mining and Case Based Reasoning*, International Journal of Computer Science Issues. 8(4): 340-349, 2011.

160. Suárez, Yanelis, *Pasos para modelar comportamientos proactivos en i* y en RUP*, Tesis de Diploma, Instituto Superior Politécnico "José Antonio Echeverría", La Habana, Cuba, 2011.

161. Sudeikat, Jan; Wolfgang Renz, *A Systemic Approach to the Validation of Self–Organizing Dynamics within MAS*, en Agent-Oriented Software Engineering IX, Lecture Notes in Computer Science, Springer Berlin Heidelberg. 5386: 31-45, 2009.

162. Talbi, El Ghazali, *Metaheuristics: From Design to Implementation*. 1st ed., John Wiley & Sons, Inc., 2009.

163. Tian, Jeff, *Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement*. 1st ed., IEEE Computer Society, New Jersey, 2005.

164. Tomassini, Marco; Leonardo Vanneschi; Philippe Collard; Manuel Clergue, *A Study of Fitness Distance Correlation as a Difficulty Measure in Genetic Programming*, Evolutionary Computation. 13(2): 213-239, 2005.

165. van der Hoek, Wiebe; Michael Wooldrige, *Multi-Agent Systems*, en Handbook of Knowledge RepresentationElsevier Science. 887-928, 2008.

166. Villanueva, Pilar, *Extrusión: Informe de vigilancia tecnológica*, Observatorio del Plástico, Reporte Técnico, 2012. Disponible en: http://www.observatorioplastico.com/detalle_publicacion.php?id_publicacion=224.

167. Wagner, Volker; Anwyn Dullaart; Anne-Katrin Bock; Axel Zweck, *The Emerging Nanomedicine Landscape*, Nature Biotechnology. 24(10): 1211-1217, 2006.

168. Wang, Hongfeng; Dingwei Wang; Shengxiang Yang, *A memetic algorithm with adaptive hill climbing strategy for dynamic optimization problems*, Soft Computing-A Fusion of Foundations, Methodologies and Applications. 13(8-9): 763-780, 2009.

169. Withall, Stephen, *Software Requirement Patterns*. O'Reilly Media Inc, 2010.

170. Wolpert, David H.; William G. Macready, *No Free Lunch Theorems for Optimization*, IEEE Transactions on Evolutionary Computation. 1(1): 67-82, 1996.

171. Wongthongtham, Pornpit; Elizabeth Chang; Tharam Dillon; Ian Sommerville, *Development of a Software Engineering Ontology for Multi-site Software Development*, IEEE Transactions on Knowledge and Data Engineering. 21(8): 1205-1217, 2009.

172. Wooldridge, Michael, *An Introduction to MultiAgent Systems*. 2nd ed., John Wiley & Sons, 2009.

173. Wooldridge, Michael; Nicholas Jennings, *Intelligent agents: Theory and Practice*, The knowledge Engineering Review. 10(2): 115-152, 1995.

174. Yin, Robert K., *Case Study Research: Desing and Methods*. 3rd ed., SAGE Publications, International Educational and Professional Publishers, 2003.

175. Yu, Eric, *Modelling Strategic Relationships for Process Reengineering*, Tesis Doctoral, University of Toronto, Toronto, Canada, 1995.

176. Yu, Eric, *Social Modeling and i**, en Conceptual Modeling: Foundations and Applications, Springer-Verlag. 2009.

177. Yu, Eric; Paolo Giorgini; Neil Maiden; John Mylopoulos, *Social Modeling for Requirements Engineering*. 1st ed., The MIT Press, 2011.

178. Zambonelli, F.; N. Jennings; M. Wooldridge, *Developing Multiagent System: The Gaia Methodology*, ACM Transactions on Software Engineering and Methodology. 12(3): 317-370, 2003.

179. Zambonelli, Franco; Nicholas Jennings; Michael Wooldridge, *Multi-Agent Systems as Computational Organizations: The Gaia Methodology*, en Agent-Oriented Methodologies, Idea Group Inc. 2005.

# *Anexo 1: Producción científica del autor sobre el tema de la tesis*

Revistas

- M. Moreno, A. Rosete, A. Simón, R. Valdés, E. Leyva, R. Socorro, J. Pina, A. García, Ingeniería de Software Orientada a Agentes: los roles y las metodologías, Revista de Ingeniería Industrial, Vol. 27, No.2-3, pp.25-28, ISSN 0238-3960, CUJAE, Ciudad de la Habana, Cuba, 2006.
- M. Moreno, A. Rosete, A. Simón, R. Valdés, E. Leyva, R. Socorro, J. Pina, A. García, Los roles y las metodologías en la Ingeniería de Software Orientada a Agentes, Revista Ingeniería al Día, Año 3, No. 2, 2006, ISSN 0718-073X, Facultad de Ciencias Físicas y Matemáticas, Universidad Central de Chile, Santiago de Chile, Chile.
- M. Moreno, Z. Fernández, M. Lorenzo, A. Rosete, M. Delgado, Y. Hadfeg. Integración de Metodologías Orientadas a Agentes, Revista Cubana de Ciencias Informáticas, Vol. 1, No 4, pp. 4-19, ISSN 1994-1536, UCI, Ciudad de la Habana, Cuba, 2007.
- M. Moreno, J. Pavón, A. Rosete, Testing in Agent Oriented Methodologies. Lecture Notes in Computer Science, Vol. 5518, pp. 138–145, Springer-Verlag Berlin Heidelberg, 2009. (Scopus)
- Y. Hadfeg, M. Moreno, M. Delgado, Propuesta de actividades de pruebas para Ingenias, Revista Cubana de Ingeniería, vol. 3, no. 3, pp. 47-56, 2012. (DOAJ)
- Mailyn Moreno, Alternán Carrasco, Alejandro Rosete, Martha D. Delgado, Patrones de Implementación para Incluir Comportamientos Proactivos. Polibits. January-June 2013, No.47, pp.73-87, 2013. (Scielo)
- Mailyn Moreno, Alternán Carrasco, Alejandro Rosete, Martha D. Delgado, Apoyo a la toma de decisiones en un Observatorio Tecnológico

incorporando proactividad. Revista de Ingeniería Industrial, Vol. 34, No. 3, 2013. (Scielo)

- M. Moreno, A. Rosete, J. Pavón, An agent based implementation of proactive S-Metaheuristics. Lecture Notes in Computer Science, Vol. 8073, pp. 1-10, 2013. (Scopus)

Eventos

- M. Moreno, A. Rosete, E. Leyva, Ingeniería de Software Orientada a Agente: Tareas y Roles en el Proceso de Desarrollo, III Taller en Desarrollo de Sistemas Multiagentes, DESMA´07, II Congreso Español de Informática 2007, ISBN 978-84-9732-613-1, Zaragoza, España, 11-18 de septiembre, 2007.
- M. Moreno, A. Rosete, Z. Fernández, Y. Hadfeg. Hacia una Metodología Integrada de Agentes con Roles, V Jornada para el Desarrollo de Grandes Aplicaciones de Red, JDARE 2008, Alicante, España, 16-17 de octubre, 2008, Actas, ISBN-13: 978-84-612-6812-2, p. 187-200.
- Y. Hadfeg, M. Moreno. Un Caso de Estudio con la Metodología Integrada de Agentes, III Taller de Informática Aplicada, V Simposio de Ingeniería Industrial, Informática y Afines, XIV Convención científica de Ingeniería y Arquitectura, La Habana, Cuba, 2-5 de Diciembre, 2008, ISBN 978-959-261-281-5.
- M. Moreno, J. Pavón, A. Rosete. New Task and Role of Persons in Ingenias, 4to Taller Internacional de Calidad en las Tecnologías de Informática y las Comunicaciones, XIII Convención Internacional Informática 2009, ISBN 978-959-286-010-0, La Habana, Cuba, febrero, 2009.
- M. Moreno, A. Rosete, J. Pavón, An agent based implementation of proactive S-Metaheuristics, International Conference on Hybrid Artificial Intelligence Systems, Salamanca, España, septiembre, 2013.
- A. Rosete, M. Moreno, Proactive selection of metaheuristics based on knowledge of previous results, Fourth International Workshop on Knowledge

Discovery, Knowledge Management and Decision Support, Eureka Workshop 2013, November 6-8, 2013, Mazatlan, Mexico. Aceptado, Proceedings book edited by Atlantis Press www.atlantis-press.com

Capítulos de libros

- L. Ceccaroni, A. Simón, A. Rosete, M. Moreno. Cognitive Agents as a Design Metaphor in Environmental-Knowledge Management. Advanced Agent-Based Environmental Management Systems, Whitestein Series in Software Agent Technologies and Autonomic Computing, U. Cortés; M. Poch (Eds.), 2009, VI, Birkhäuser Verlag Basel/Switzerland, ISBN 978-3-7643-8897-3, p. 139-169.

Tesis de Maestría

| **Autores** | **Datos de la Tesis** |
|---|---|
| Mailyn Moreno Espino | Metodologías orientadas a agentes: un estudio comparativo, Tesis de Maestría, CUJAE, julio de 2006, Autor: M. Moreno, Tutores: Dr. Alejandro Rosete, Dra. Ailyn Febles. |
| Yahima Hadfeg Fernádez | Ingenias: Incorporando Análisis de Requisitos, Cujae, julio 2011, Tutor: MSc. Mailyn Moreno Espino. |

Tesis de Diplomado

| **Autores** | **Datos de la Tesis** |
|---|---|
| Yahima Hadfeg Fernádez | Transformación de artefactos i* a UML 2.0: Propuesta Inicial, cujae, julio 2009, Tutor: MSc. Mailyn Moreno Espino. |
| Omarys García Fernández | Las Pruebas en los Sistemas Agentes: Una aproximación inicial, cujae, diciembre 2007, Tutores: Dr. Alejandro Rosete Suárez y MSc. Mailyn Moreno Espino. |

Tesis de Diploma

| **Autores** | **Datos de la Tesis** |
|---|---|
| Zaili Rebeca Fernández Vega y Mary Leidy Lorenzo González | Propuesta de Unificación e Integración de Metodologías Orientadas a Agentes, Cujae, Junio 2007, Tutores: MSc. Mailyn Moreno Espino y Dr. Alejandro Rosete Suárez. |
| Yahima Hadfeg Fernández | Metodología Orientada Agente Integrada: Desarrollo de un Caos de Estudio, Cujae, junio 2008, Tutores: MSc. Mailyn Moreno Espino, Dr. Alejandro Rosete |

| | |
|---|---|
| | Suárez y Lic. Lourdes Fenández Rivas. |
| Roger Pérez García | Transformación de modelos i* a modelos UML 2.0: Propuesta inicial, Cujae, Octubre 2008, Tutores: MSc. Mailyn Moreno Espino e Ing. Yahima Hadfeg Fernández. |
| Luz Marlis Haití Sanamé | Módulo de Control de Asistencia del GREHU: Una Visión desde la Ingeniería de Requisitos, Cujae, junio 2009, Tutores: MSc. Mailyn Moreno Espino e Ing. Yahima Hadfeg Fernández. |
| Leysandra Horsford Morales | Aplicación de Técnica de Ingeniería de Requisitos al Subsistema Control de Registro de Estudiantes, Cujae, junio 2009, Tutores: MSc. Mailyn Moreno Espino e Ing. Yahima Hadfeg Fernández. |
| Jorge Lázaro López Rodríguez | Módulo de Configuración del GREHUCORP. Una Perspectiva desde la Ingeniería de Requisitos, Cujae, junio 2009, Tutores: MSc. Mailyn Moreno Espino e Ing. Yahima Hadfeg Fernández. |
| Kinane Sleiman Domloje | Comparación del Análisis de Requisitos en i* y RUP del Módulo Control de Asistencias y Notas Parciales, Cujae, junio 2010, Tutores: MSc. Mailyn Moreno Espino e Ing. Yahima Hadfeg Fernández. |
| Perla Beatriz Fernández Oliva | Comparación del Análisis de Requisitos en i* y RUP en la Intranet de Informática, Cujae, junio 2010, Tutores: MSc. Mailyn Moreno Espino e Ing. Yahima Hadfeg Fernández. |
| Hermes Joaquín Vaillant Vera | Ingenias: Tareas de Pruebas, Cujae, junio 2010, Tutores: MSc. Mailyn Moreno Espino e Ing.Omarys García Fernández. |
| Ailín Valdés Torres y Leandro Serrano Abad | Propuesta de Arquitectura de Sistema Multi-Agentes que Responda a los Intereses de un Observatorio Tecnológico, Cujae, junio 2010, Tutores: MSc. Mailyn Moreno Espino e Ing. Alternán Carrasco Bustamente. |
| Yanelis Suárez Fraga | Pasos para modelar comportamientos proactivos en i* y en RUP, Cujae, junio 2011, Tutor: MSc. Mailyn Moreno Espino. |
| Dorys Perdomo Pérez | Tropos I* conversión de artefactos a Ingenias, Cujae, junio 2011, Tutor: MSc. Mailyn Moreno Espino e Ing. Yahima Hadfeg Fernández. |
| Elaine Castillas Imas | Patrones para incorporar proactividad, Cujae, julio 2012, Tutores: MSc. Mailyn Moreno Espino e Ing. Alternán Carrasco Bustamente. |
| Yasser Juan de la Flor | Herramienta de simulación basada en agentes a partir de la captura de requisitos en i*, Cujae, julio 2013, Tutores: MSc. Mailyn Moreno Espino e Ing. Alternán Carrasco Bustamente. |
| Alejandro Alonso | Controlador de patrones para JADE, Cujae, julio |

| | |
|---|---|
| Fernández | 2013, Tutores: MSc. Mailyn Moreno Espino e Ing. Alternán Carrasco Bustamente. |
| Dainiel Miranda Chavez y Taysir Bachir Taha Pérez | Propuesta de transformación de los requisitos tempranos y tardíos de i* al modelo del negocio de RUP, Cujae, julio 2013, Tutores: MSc. Mailyn Moreno Espino. |

# *Anexo 2: Conceptos básicos de i**

Conceptos Básicos

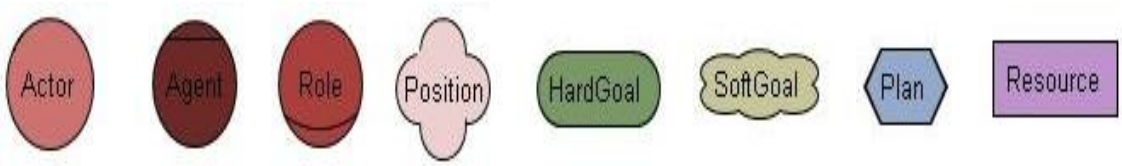



Figura 1: Notación gráfica de los elementos básicos de i* en el Taom4E.

**Actor:** es una entidad que tiene metas estratégicas e intenciones dentro del sistema o dentro del conjunto organizacional.

**Rol (*Role*):** es una caracterización abstracta de la conducta de un actor dentro de algunos contextos especializados o dominios. Sus características son fácilmente transferibles a otros actores sociales.

**Agente (*Agent*):** es un actor con manifestaciones físicas concretas como las de un humano. Es utilizado el término de agente en lugar de persona para generalizar su utilización.

**Posición (*Position*):** es una abstracción intermedia entre un rol y un agente. Se puede decir que un agente ocupa una posición y una posición cubre un rol.

**Meta fuerte/Meta suave (*hardgoal*)/(*softgoal)*:** estas metas representan los intereses estratégicos de un actor. Los *hardgoals*se distinguen de los *softgoals*porque en los segundos no existe un criterio claro para definir si ellos han sido satisfechos.

**Plan:** representa una forma de hacer algo en un nivel abstracto. La ejecución del plan puede ser una manera de satisfacer un *hardgoal* o un *softgoal*.

**Recurso (*Resource*):** representa una entidad física o de información.

El marco de trabajo I* que utiliza Tropos está formado por dos diagramas que nos permiten modelar el ambiente organizacional con un enfoque basado en metas y dependencias:

Diagrama de actores. Es una representación gráfica donde se muestran los actores y sus metas, y las dependencias entre los actores.

Diagrama de metas. Es una representación gráfica donde se analizan con mayor detalle las metas, planes y dependencias de cada actor.

**Modelo de Dependencia Estratégica**

El Modelo de Dependencia Estratégica (SD) consiste en un *set* de nodos y relaciones, cada nodo representa un actor y cada relación entre dos actores indica que un actor (*depender*) depende del otro (*dependee*) en algo para que el anterior pueda conseguir una meta un recurso o un objetivo etc., como se ilustra en la figura 2. El objeto del cual se trata una relación es el *Dependum*.

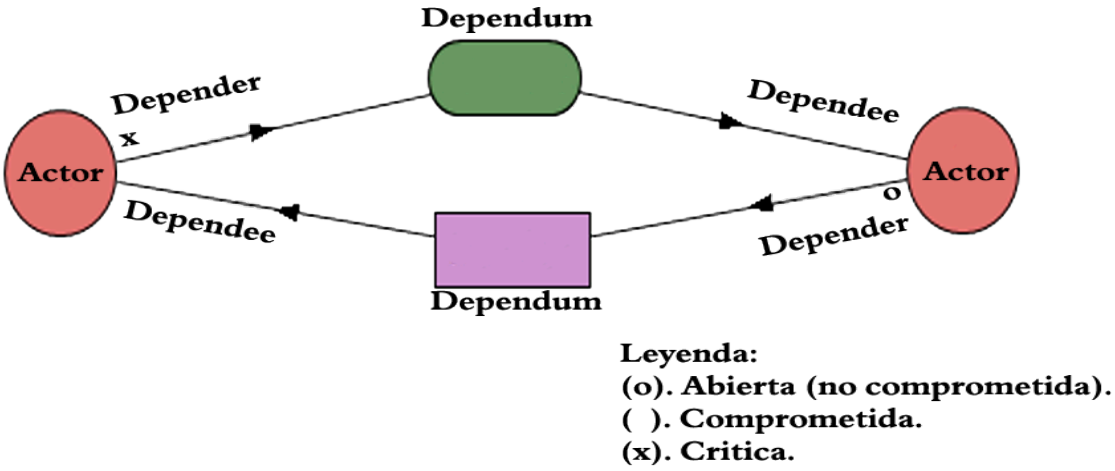



Figura 2: Modelo de Dependencia Estratégica (SD)

La dependencia es intencional si el *dependum* es de alguna manera relacionado a la meta o a un deseo del depender. Podemos distinguir cuatro tipos de dependencias basado en el tipo del *dependum*: Recurso (*Resource Dependency*), Meta (*Goal Dependency*), Tarea (*Task Dependency*) y Meta-Suave (*Softgoal Dependency*), esto se muestra en la figura 3. El modelo permite definir la importancia de la dependencia para cada uno de los actores que intervienen en ella, sea crítica (*critical*), comprometida (*commited*) o abierta (open).

**Recurso (*ResourceDependency*):** el *depender* depende del *dependee* para garantizar la disponibilidad de una entidad, esto le permitiría al depender usar esta entidad como un recurso.

**Meta (*GoalDependency*):** el *depender* depende del *dependee* para devolver un estado en el entorno. El*depende* está libre de elegir como hacerlo.

**Tarea (*TaskDependency*):** el *depender* depende del *dependee* para realizar una actividad (tarea). Una dependencia de tarea especifica cómo se tiene que realizar la actividad pero no porque.

**Meta-Suave (*SoftgoalDependency*):** el *depender* depende del *dependee* para realizar una meta-suave, una meta-suave se especifica en términos de los métodos que se eligen en el curso de perseguir la meta. La diferencia entre meta-suave y meta es que la condición buscada están elaborada en el transcurso de la tarea.

El sentido de la dependencia es simbolizado por una flecha que aparece en los caminos de la red. Por ejemplo: un actor (*depender*) va a depender del otro (*dependee*) cuando la flecha apunta hacia el segundo.

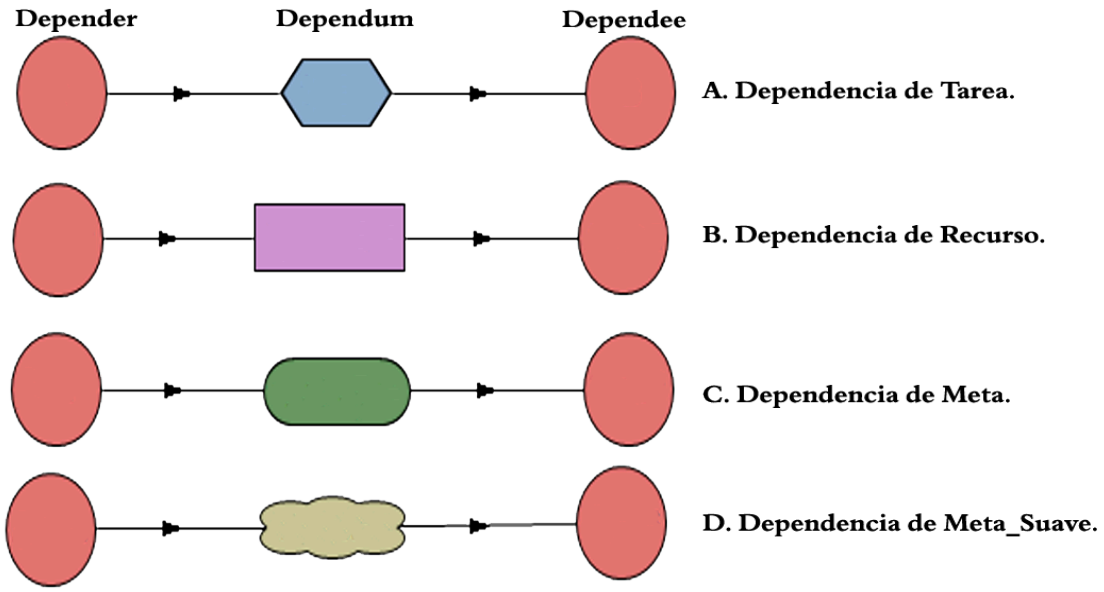



Figura 3: Tipos de Dependencias.

**Modelo de razones estratégicas**

El modelo de razones estratégicas (SR) se centra más detalladamente en los actores por dentro, con el propósito de poder modelar las relaciones internas de cada actor. El SR es utilizado para describir y soportar todo el nivel de razonamiento que tiene cada actor acerca de sus relaciones con los demás actores, constituye la base fundamental, para el refinamiento del modelo SD y añade cierto nivel de razonamiento. Está constituido por una gráfica que presenta un conjunto de nodos y relaciones, que expresan todas aquellas relaciones que hay detrás de los procesos tal y como se muestra en la figura 4.

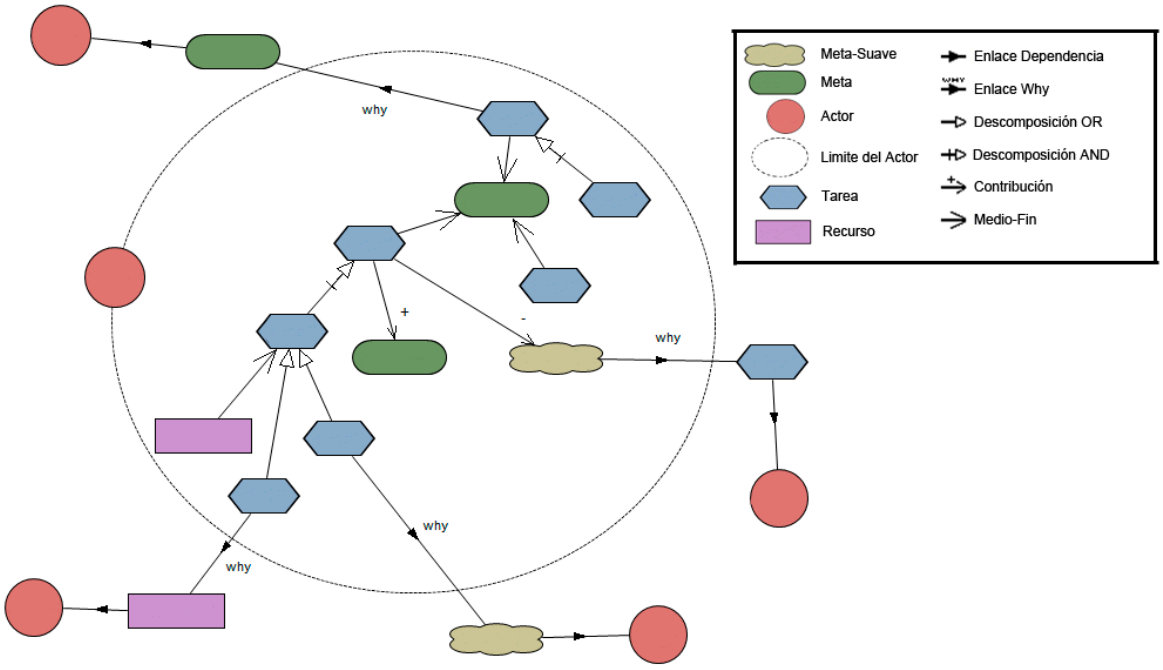



Figura 4: Vista general de un Modelo de Razones Estratégico.

En los modelos SR existe dos tipos de relaciones las cuales se describen a continuación:

**Relación medio-fin (*means-end*):** esta relación se establece entre dos elementos intencionales de un mismo actor, donde generalmente uno de los medios es una tarea que contribuye a la realización de un fin. Cuando hay existencia de varios medios para el cumplimiento de un fin determinado, se presencia una relación *OR* que indica las distintas maneras de lograr un fin. En los modelos *SR* este tipo de relación permite comprender el “porque” un actor realiza una tarea, persigue alguna meta, necesita algún recurso o desea obtener una meta-suave. Un ejemplo

de las relaciones de este tipo que pueden ser posible se describe a continuación y se ilustran en la figura 5.

**Relación Meta - Tarea:** la meta especifica el fin y la tarea el medio. Esta relación expresa el "como" la tarea contribuye a la realización de una meta. Para lograr el cumplimiento de la meta es necesaria la realización de la tarea.

**Relación Meta-suave - Tarea:** el fin es especificado mediante la meta-suave y el medio como la tarea. La relación indica que la realización de la tarea va a ser imprescindible para que la meta-suave se alcance.

**Relación Recurso - Tarea:** el fin se especifica mediante el recurso y el medio a través de la tarea. Esta relación expresa que para poder disponer de un recurso hace falta realizar una tarea.

**Relación Meta-suave - Meta-suave:** permite el desarrollo de organizaciones medio-fin en metas-suaves, hasta que finalmente algunas metas-suaves, sean Tratadas como tareas. En las relaciones como estas donde una Meta Suave aparece como fin, se indica si la contribución es positiva o negativa. En la notación gráfica la punta de la flecha, representa el fin.

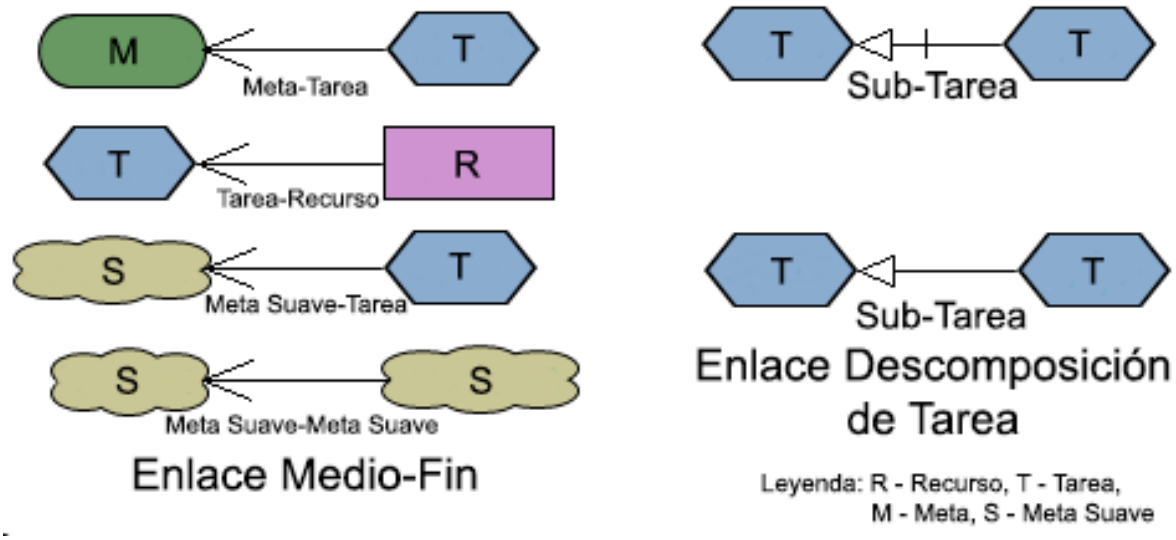



Figura 5: Tipos de enlaces.

**Relación de Descomposición de Tareas (*task-descomposition*):** en esta relación una tarea queda descompuesta en determinados elementos intencionales

de cualquier tipo, cuando se distingue la presencia de varios elementos, hay existencia de una relación *AND* entre ellos, para de esta forma completar con restricciones (*constraints*) para definir esta relación. Una tarea en este tipo de relación puede estar descompuesta en sub-metas, sub-tarea, recursos y en metas-suaves tal y como se muestra en la figura 5.

# *Anexo 3: Clases de la operación Guardar Trazas*

Las clases que permiten el mecanismo de guardar trazas se describen a continuación:

*LogController*: esta clase controla todo el proceso de salva de log, decide en que ficheros se deben almacenar y realiza un análisis de los log que no fueron almacenados un primer momento. Los log se almacenan en el mismo momento que se reciben los mensajes.

*LogSend*: contiene la información de los mensajes que se envían los agentes tales como Id, emisor y la hora que se envió, estos van a ser almacenados en forma de log.

*LogReceive*: contiene la información de los mensajes recibidos tales como Id, receptor, contenido, tipo, la hora en que se recibió y performativa del mensaje, estos van a ser almacenados en forma de log.

*LogXES*: esta clase es la encargada de crear los ficheros con extensión .xes y formato XML, que contienen los log.

En la Figura 1 se muestra el diagrama de clases del mecanismo de obtención los ficheros XES

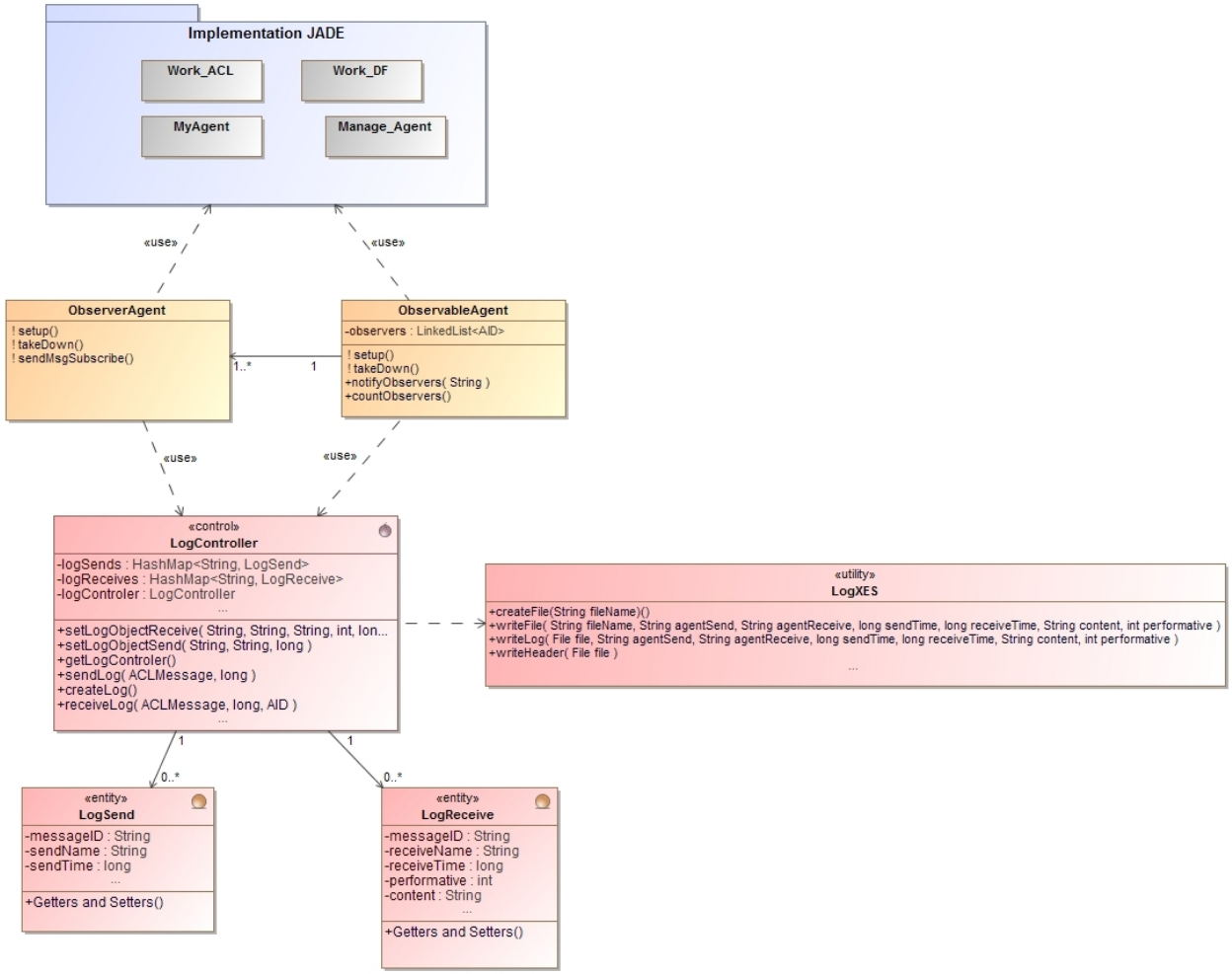



Figura 1: Diagrama de clases del mecanismo de obtención de ficheros XES

# *Anexo 4: Simulación del Observatorio*

En esta sección se presentan los resultados obtenidos de la simulación del OT del CITI a partir de la captura de requisitos en i*.

Configuración y objetivos perseguidos

En la Tabla 1 se muestra la configuración utilizada para esta simulación. Para la simulación se seleccionaron todos los actores presenten en los requisitos del OT. También fueron establecidos los tiempos de respuesta para cada tipo de agente, así como la cantidad de instancias de cada uno.

Tabla 1 Configuración de la simulación del OT.

| Actor | Cantidad | Tiempo (milisegundos) |
|---|---|---|
| Investigador | 67 | 10000 |
| Agente Personal | 67 | 15000 |
| Confianza | 1 | 10000 |
| Analista | 1 | 30000 |
| Fuente Datos | 5 | 60000 |

Para esta simulación se trazaron 3 objetivos:

- Comprobar que siempre que un investigador inicia un pedido exista un agente personal que satisfaga dicho pedido, y en caso de no ser así, buscar cuáles serían las posibles causas.
- Comprobar consumo de hardware, desde el punto de vista del uso del microprocesador dedicado al OT en el servidor. Para este objetivo se estableció como directiva que el 100% del microprocesador se usa cuando todos los agentes Fuente de Datos configurados se encuentran en el estado de “procesando respuesta”.
- Conocer el tiempo que pudiera estar esperando un Agente Personal para que el agente Analista responda a su solicitud.

Para responder a estos objetivos se realizaron 5 iteraciones con la misma configuración, comprobando y evaluando los resultados obtenidos. Debido a las características antes mencionadas de los agentes, estas simulaciones van a reflejar ***n*** posibles casos que se puedan dar durante el proceso.

Resultados obtenidos

Persiguiendo el objetivo de comprobar que siempre que un investigador inicie un pedido exista un agente personal que satisfaga dicho pedido y en caso de no ser así, buscar cuales serían las posibles causas, se obtuvieron los siguientes resultados comprobando las trazas de la simulación en el fichero csv, así como los gráficos de línea de tiempo generados por la herramienta.

Durante la primera iteración hubo 582 pedidos de los investigadores que estuvieron en espera a ser atendidos. Esto se debe a que en esta iteración 51 pedidos del Agente Personal al Analista y 8 pedidos del Agente Personal al agente Fuente de Datos se hallaron en espera de ser atendidos. Estas situaciones trajeron consigo que de los 67 Agentes Personales simulados, de 28 a 36 se encontraron en el estado de "esperando respuesta" durante el tiempo de simulación, lo cual se puede apreciar en el gráfico de la Figura 1.

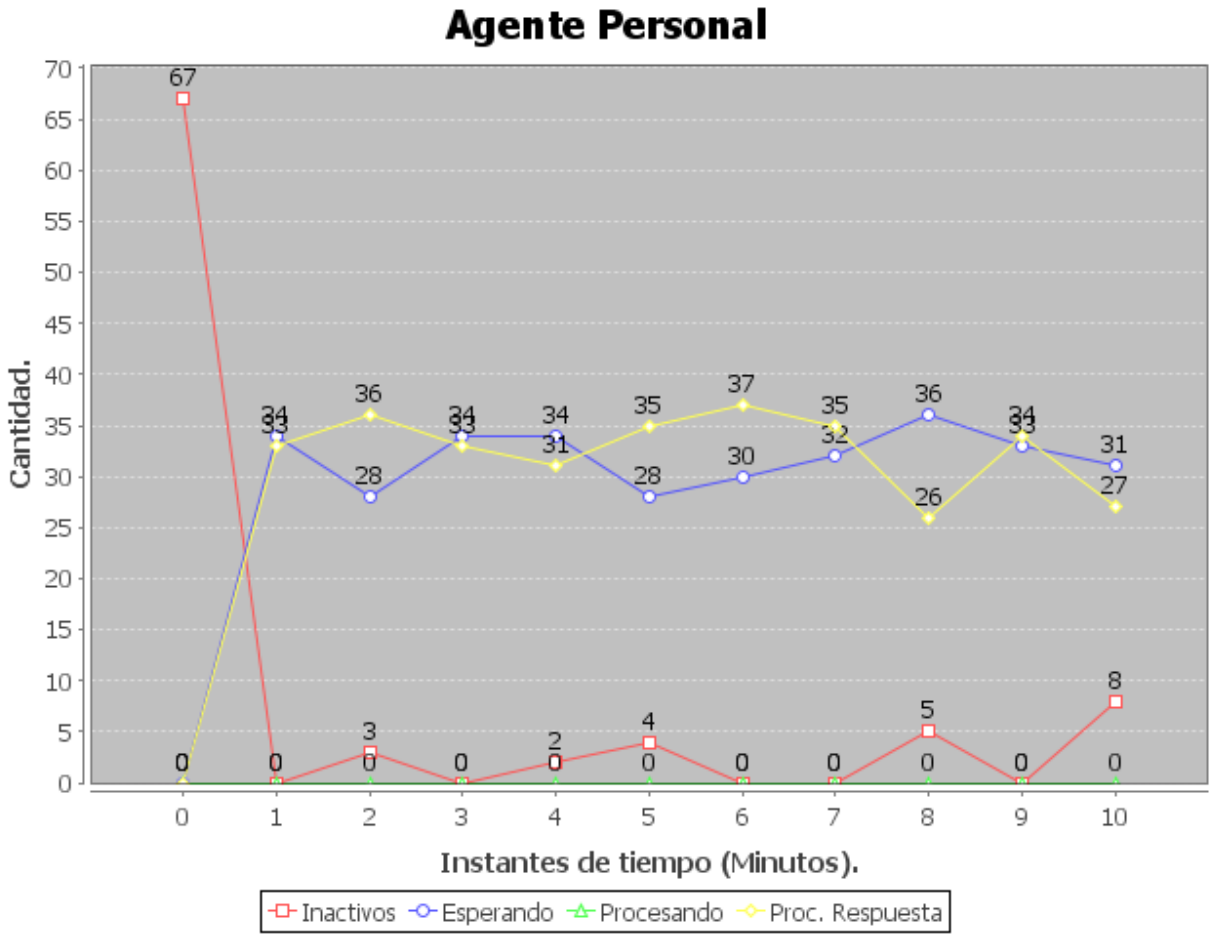



Figura 1: Comportamiento de los Agentes Personales durante la iteración 1 de la simulación del OT del CITI

Durante la segunda iteración hubo 1469 pedidos de los investigadores que estuvieron en espera a ser atendidos. Esto se debe a que en esta iteración 38 pedidos del Agente Personal al Analista y 7 pedidos del Agente Personal al agente Fuente de Datos se hallaron en espera de ser atendidos. Esto provocó que de los 67 Agentes Personales simulados, de 17 a 28 se encontraron en el estado de "esperando respuesta" durante el tiempo de simulación, lo cual se puede apreciar en el gráfico de la Figura 2.

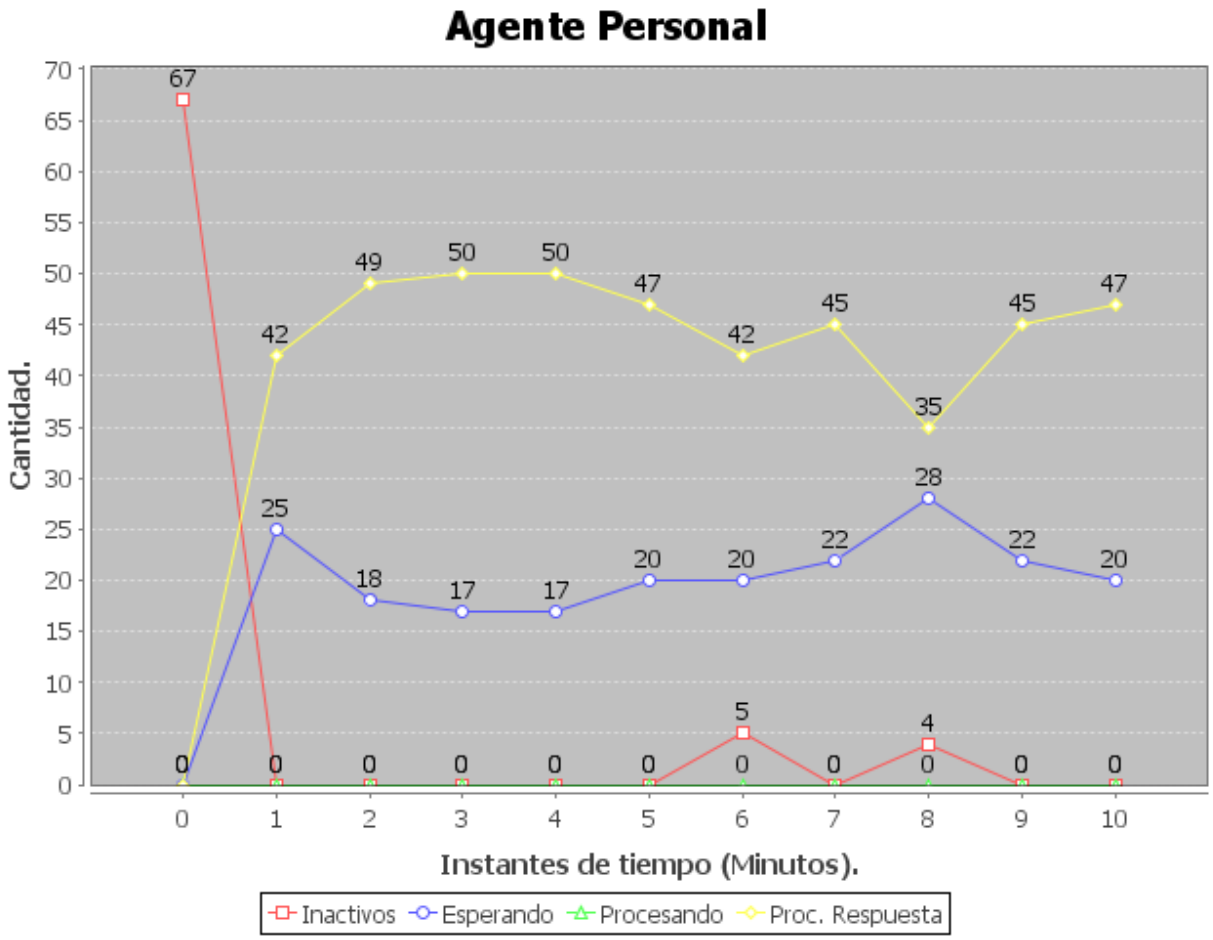



Figura 2: Comportamiento de los Agentes Personales durante la iteración 2 de la simulación del OT del CITI.

Durante la tercera iteración no hubo pedidos de los investigadores que estuvieran en espera a ser atendidos. Esto se debe fundamentalmente a que en esta simulación no hubo muchos pedidos por parte de los investigadores a los agentes personales lo cual se puede apreciar en la Figura 7, ya que en esta iteración 51 pedidos del Agente Personal al Analista y 8 pedidos del Agente Personal al agente

Fuente de Datos se hallaron en espera de ser atendidos. Todo esto trajo consigo que de los 67 Agentes Personales simulados, de 10 a 35 se encontraron en el estado de “esperando respuesta” durante el tiempo de simulación, lo cual se puede apreciar en el gráfico de la Figura 3.

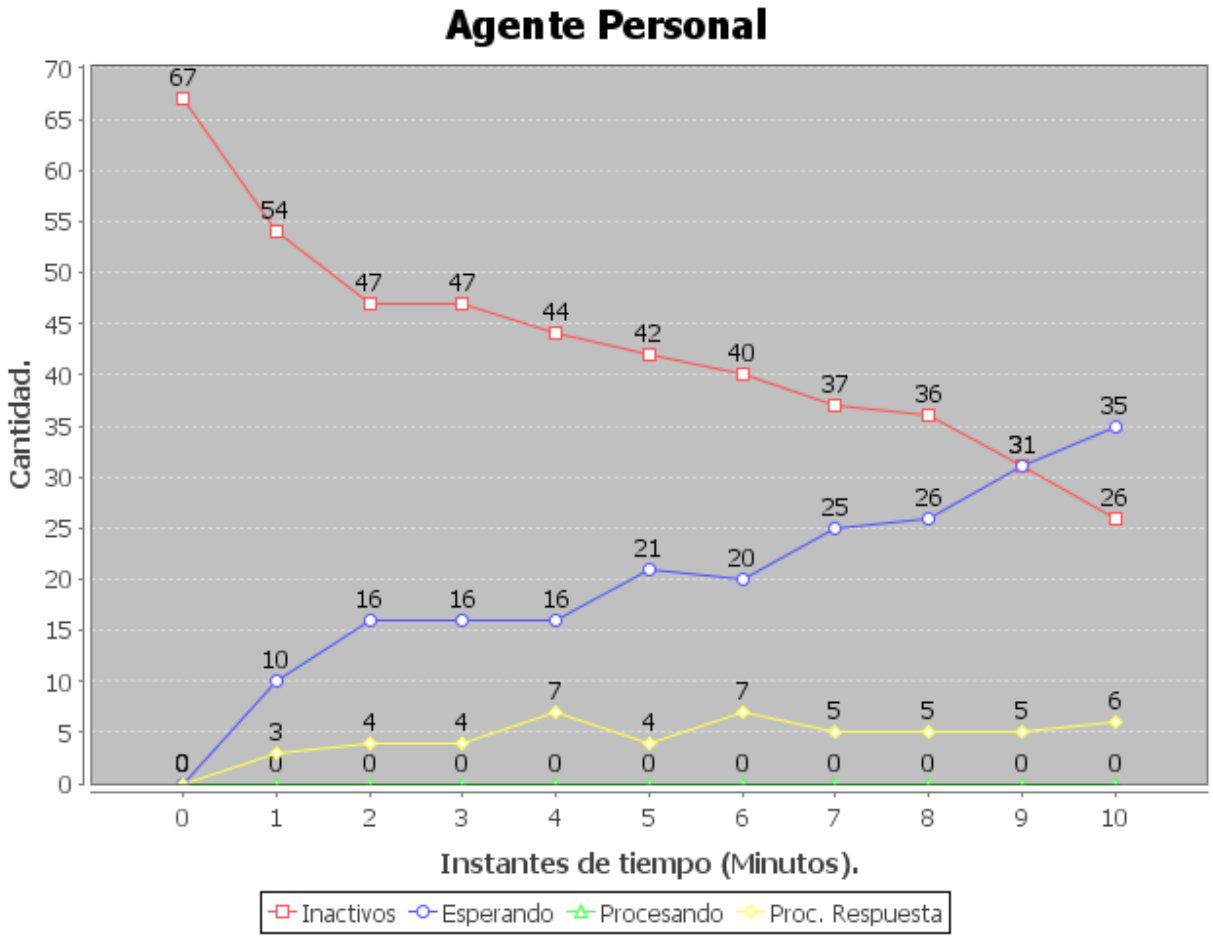



Figura 3: Comportamiento de los Agentes Personales durante la iteración 3 de la simulación del OT del CITI.

Durante la cuarta iteración no hubo pedidos de los investigadores que estuvieran en espera a ser atendidos. Esto se debe fundamentalmente a que en esta simulación no hubo muchos pedidos por parte de los investigadores a los agentes personales, lo cual se puede apreciar en la Figura 7, ya que en esta iteración 62 pedidos del Agente Personal al Analista se hallaron en espera, esto trajo consigo que de los 67 Agentes Personales simulados de 8 a 48 se encontraron en el estado de “esperando respuesta” durante el tiempo de simulación, lo cual se puede apreciar en el gráfico de la Figura 4.

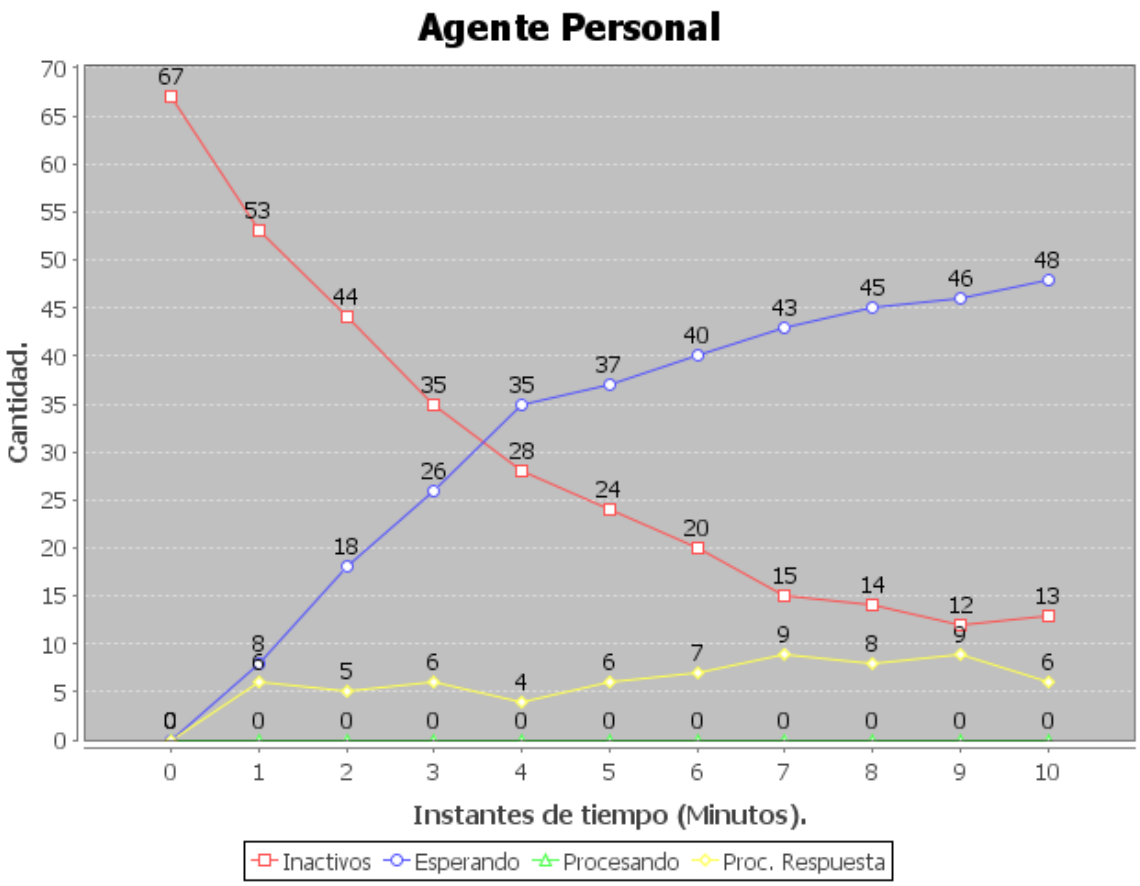



Figura 4: Comportamiento de los Agentes Personales durante la iteración 4 de la simulación del OT del CITI.

Durante la quinta iteración no hubo pedidos de los investigadores que estuvieran en espera a ser atendidos. Esto se debe fundamentalmente a que en esta simulación no hubo muchos pedidos por parte de los investigadores a los agentes personales, lo cual se puede apreciar en la Figura 7, ya que en esta iteración 65 pedidos del Agente Personal al Analista y 18 pedidos del Agente Personal al agente Fuente de Datos, se hallaron en espera de ser atendidos. Todo esto trajo consigo que de los 67 Agentes Personales simulados, de 20 a 47 se encontraron en el estado de “esperando respuesta” durante el tiempo de simulación, lo cual se puede apreciar en el gráfico de la Figura 5.

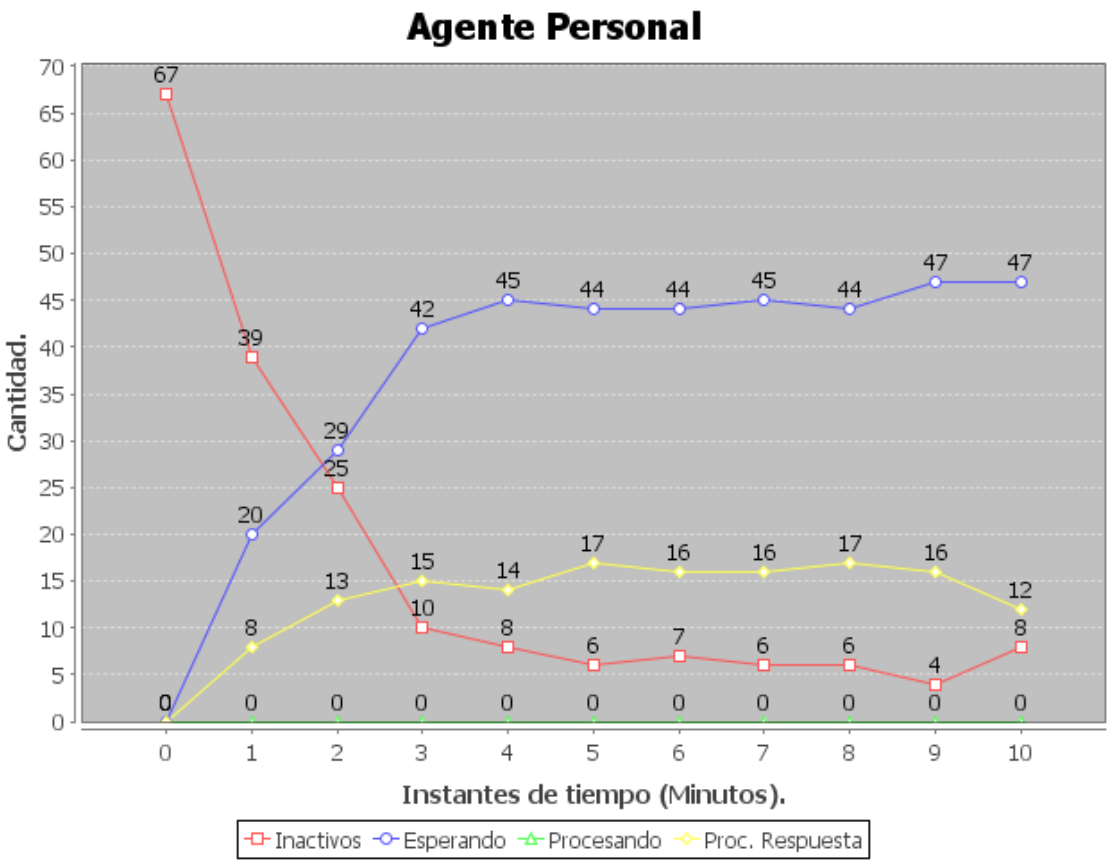



Figura 5: Comportamiento de los Agentes Personales durante la iteración 5 de la simulación del OT del CITI.

En la Figura 6 se refleja el comportamiento de la cola de trabajo de los agentes personales durante las 5 iteraciones en varios instantes de tiempo.

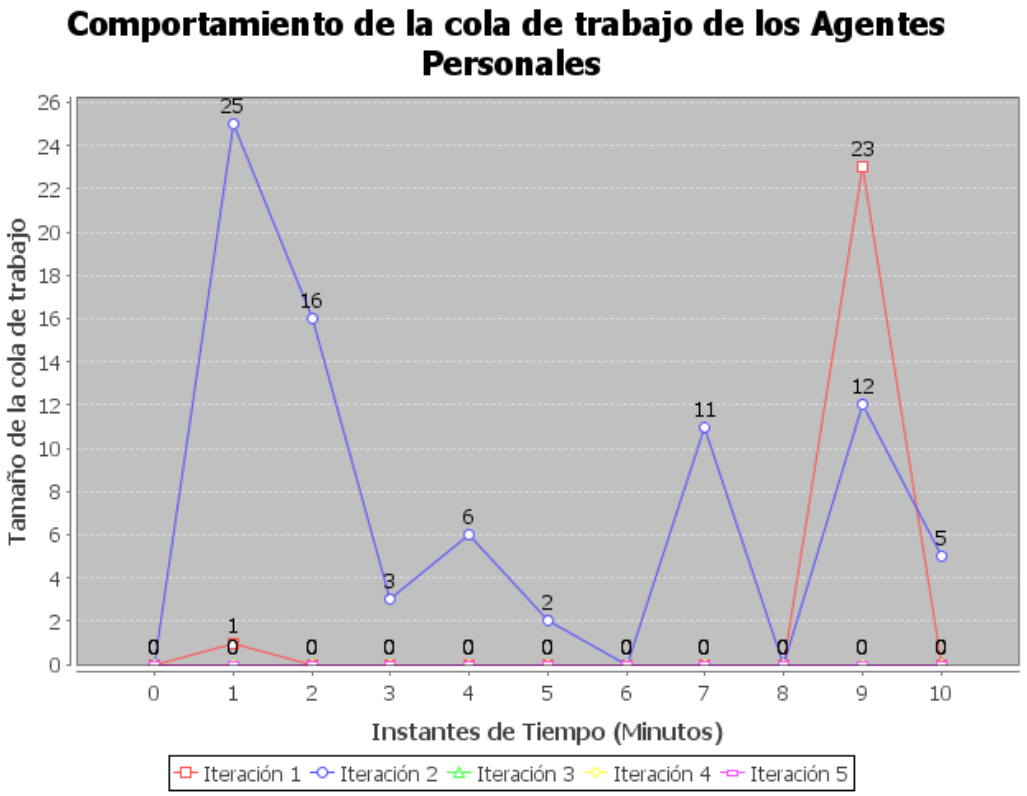



Figura 6: Comportamiento de la cola de trabajo de los Agentes Personales en las 5 iteraciones para varios instantes de tiempo.

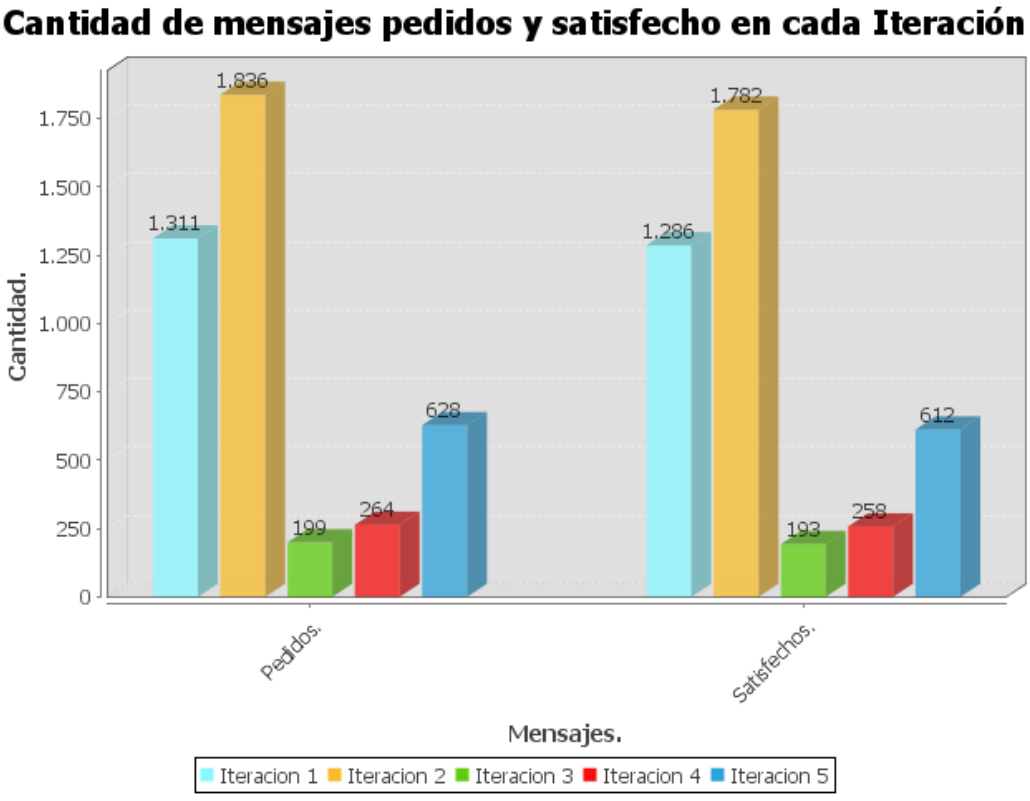


Figura 7: Cantidad de mensajes pedidos a los Agentes Personales y satisfechos por los Agentes Personales durante las 5 iteraciones realizadas.

Persiguiendo el objetivo de comprobar el consumo de hardware, desde el punto de vista del uso del microprocesador dedicado al OT en el servidor, se estableció como directiva que el 100% del microprocesador se usa cuando todos los agentes Fuente de Datos configurados se encuentran en el estado de "procesando respuesta". Se obtuvieron los siguientes resultados:

Como se puede apreciar en la Figura 8, el 80% de los resultados de las iteraciones coinciden en que una vez iniciado el OT hay un alto consumo del microprocesador dedicado a esta herramienta en el servidor, ya que todos los agentes Fuentes de Datos se encontraron en el estado de "procesando respuesta". También se puede apreciar en la Figura 8 que una vez que el OT empezó a avanzar en el tiempo de ejecución, disminuyó el uso del microprocesador entre el 40% y el 60%.

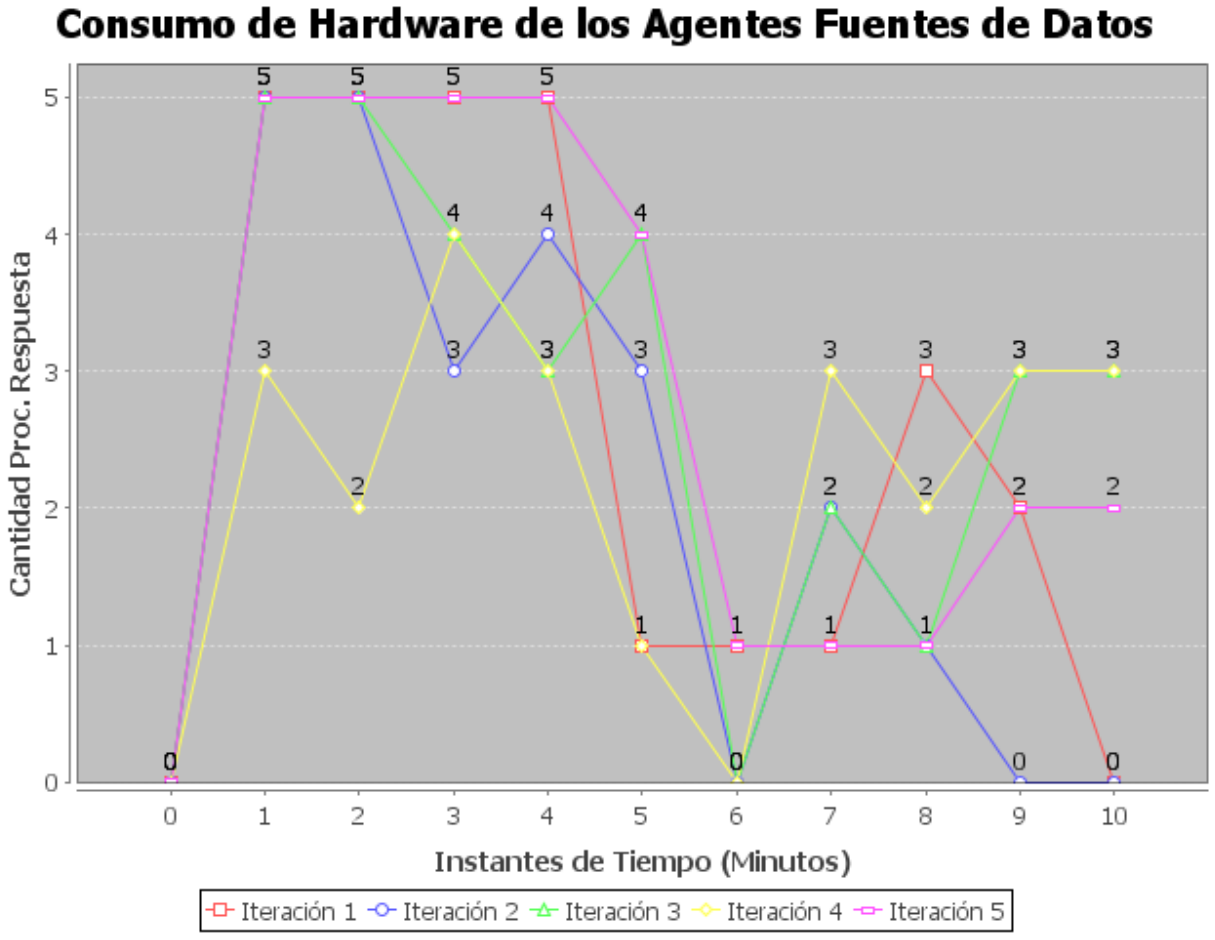


Figura 8: Cantidad de agentes Fuentes de Datos procesando respuesta en cada instante de tiempo.

Persiguiendo el objetivo de conocer el tiempo posible de espera de un Agente Personal para que el agente Analista responda a su solicitud, se obtuvieron los siguientes resultados, comprobando en las trazas de la simulación del fichero csv el tiempo transcurrido desde que un pedido entra en la cola de espera y es satisfecho. Los resultados se muestran en la Tabla 2.

Tabla 2: Promedio de tiempo de respuesta de un agente Analista a un Agente Personal.

| Iteraciones | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Promedio | 5’:03” | 4’:44” | 3’:45” | 4’:26” | 5’:06” |

# *Anexo 5: Líneas de código con y sin patrón Implementation_JADE*

**Líneas de código sin utilizar el patrón *Implementation_JADE*.**

```
*Init_Platform.init(true, null, false);
    agentContainers = new HashMap<String, AgentContainer>();
    if (_port == null || _port.isEmpty() || _port.equals(""))
        port = "1099";
    else
        port = _port;
    new Thread(){
        public void run(){
            ArrayList<String> arr = new ArrayList<String>();
            if (_showGUI)
                arr.add("-gui");
            arr.add("-port");
            arr.add(port);
            if (_autocleanupDF){
                arr.add("-jade_domain_df_autocleanup");
                arr.add("true");}
            String[] args = new String[arr.size()];
            for (int i = 0; i < args.length; i++)
                args[i] = arr.get(i);
            jade.Boot.main(args);}
            }.start();
    rt = jade.core.Runtime.instance();
    rt.setCloseVM(true);
    try{Thread.sleep(1000);}
    catch (InterruptedException e){e.printStackTrace();}
    boolean notConnected = true;
    while (notConnected){
        try{new Socket("localhost", Integer.parseInt(port));
            notConnected = false;}
        catch (NumberFormatException e){e.printStackTrace();}
        catch (UnknownHostException e){e.printStackTrace();}
        catch (IOException e){
            System.err.println("Error: " + e.getMessage());
            System.err.println("Reconnecting in one second");
            try{Thread.sleep(1000);}
            catch (InterruptedException e1){e1.printStackTrace();}}}
*Init_Platform.createAgentContainer("Server_Agent");
    Profile profile = new ProfileImpl();
```

```
    profile.setParameter(Profile.CONTAINER_NAME, containerName);
    AgentContainer agents_Container = rt.createAgentContainer(profile);
    agentContainers.put(containerName, agents_Container);
*Init_Platform.addAgent_ToContainer("Server_Agent", "Server_Agent", "test.client_server.Server_Agent", new Object[0]);
    new Thread(){
        public void run(){
            try{AgentContainer agents_Container = agentContainers.get(containerName);
                AgentController agent = agents_Container.createNewAgent(agentName, agentClass, agentArgs);
                System.out.println("Starting up " + agentName + "...");
                agent.start();}
            catch (Exception e){e.printStackTrace();}}
    }.start();
    *Work_DF.register_ByType_DF(this, "Tipo_Agente_Servidor");
        boolean register = false;
        DFAgentDescription[] df_ad_old = get_AgentDescription_ByName(agent, agent.getName());
        if (df_ad_old.length == 0){
            DFAgentDescription df_ad_new = new DFAgentDescription();
            ServiceDescription sd = new ServiceDescription();
            sd.setType(type);
            sd.setName(agent.getName());
            df_ad_new.addServices(sd);
            df_ad_new.setName(agent.getAID());
            try{DFService.register(agent, df_ad_new);
                register = true;}
            catch (FIPAException e){
                agent.doDelete();
                register = false;}}
        else{DFAgentDescription df_ad_new = df_ad_old[0];
            ServiceDescription sd = new ServiceDescription();
            sd.setType(type);
            sd.setName(agent.getName());
            df_ad_new.addServices(sd);
            df_ad_new.setName(agent.getAID());
            try{DFService.deregister(agent);
                DFService.register(agent, df_ad_new);
                register = true;}
            catch (FIPAException e){
                agent.doDelete();
```

```
77                  register = false;}}
78              return register;
79          *addBehaviour(new ProcessMsg_Server(this, null));
80              private static final long serialVersionUID = 1L;
81              Agent agent;
82              private MessageTemplate mt;
83              public Behaviour_Receive_Msg(Agent _agent, MessageTemplate _mt)
84              {agent = _agent;
85                  mt = _mt;}
86              public void action(){
87                  ACLMessage msg;
88                  if (mt == null)
89                      msg = agent.receive();
90                  else
91                      msg = agent.receive(mt);
92                  if (msg != null){
93                      processMessage(msg);}
94                  else block();}
95              *System.out.println("Soy el Agente Servidor, el Agente Cliente dice: " + msg.getContent());
96              *System.out.println("soy el Agente Servidor: mandando respuesta");
97              *ACLMessage reply = msg.createReply();
98              *reply.setContent("me alegro de que estes vivo");
99              *myAgent.send(reply);
100 *Join_Platform.init("10.9.12.113");
101     hostName = _hostName;
102     rt = jade.core.Runtime.instance();
103     agentContainers = new HashMap<String, jade.wrapper.AgentContainer>();
104 *Join_Platform.createAgentContainer("Client_Agent", "1099");
105     Profile profile = new ProfileImpl();
106     profile.setParameter(Profile.CONTAINER_NAME, containerName);
107     profile.setParameter(Profile.MAIN_HOST, hostName);
108     profile.setParameter(Profile.MAIN_PORT, _port);
109     jade.wrapper.AgentContainer agents_Container = rt.createAgentContainer(profile);
110     agentContainers.put(containerName, agents_Container);
111 *Join_Platform.addAgent_ToContainer("Client_Agent", "Client_Agent", "test.client_server.Client_Agent", new Object[0]);
112     new Thread(){
113         public void run(){
114             try{jade.wrapper.AgentContainer agents_Container = agentContainers.get(containerName);
```

```
        try{jade.wrapper.AgentContainer agents_Container = agentContainers.get(containerName);
            jade.wrapper.AgentController agent = agents_Container.createNewAgent(agentName, agentClass, agentArgs);
            System.out.println("Starting up " + agentName + "...");
            agent.start();}
        catch (Exception e){e.printStackTrace();}}
}.start();
*addBehaviour(new ProcessMsg_Client(this, null));
    private static final long serialVersionUID = 1L;
    Agent agent;
    private MessageTemplate mt;
    public Behaviour_Receive_Msg(Agent _agent, MessageTemplate _mt)
    {agent = _agent;
        mt = _mt;}
    public void action(){
        ACLMessage msg;
        if (mt == null)
            msg = agent.receive();
        else
            msg = agent.receive(mt);
        if (msg != null){
            processMessage(msg);}
        else block();}
    *System.out.println("Soy el Agente Cliente, el Agente Servidor se alegra de que yo este vivo");
*System.out.println("Soy el Agente Cliente: mandando mensaje");
*Work_ACL.sendMessage_ByType(this, "Tipo_Agente_Servidor", "ya estoy vivo");
    AID aid_receiver = Work_DF.get_AIDAgent_ByType(sender, type);
        DFAgentDescription[] result = Work_DF.get_AgentDescription_ByType(agent, type);
            DFAgentDescription df_ad = new DFAgentDescription();
            ServiceDescription sd = new ServiceDescription();
            sd.setType(type);
            df_ad.addServices(sd);
            DFAgentDescription[] result = new DFAgentDescription[0];
            try{result = DFService.search(agent, df_ad);}
            catch (Exception e){e.printStackTrace();}
        AID aidAgent = null;
        int length = result.length;
        if (result != null && length > 0)
            aidAgent = result[0].getName();
    ACLMessage msg = createACLMessage(-1, sender, aid_receiver, content);
```

```
153             ACLMessage msg = null;
154             if (performative == -1)
155                 msg = new ACLMessage(ACLMessage.UNKNOWN);
156             else
157                 msg = new ACLMessage(performative);
158             msg.setSender(sender.getAID());
159             msg.setLanguage(language);
160             if (receiver != null)
161                 msg.addReceiver(receiver);
162             else{int length = receivers.length;
163                 if (receivers != null && length > 0)
164                     for (int i = 0; i < length; i++)
165                         msg.addReceiver(receivers[i]);}
166
167             if (content != null && content != ""){
168                 msg.setContent(content);}
169             else{
170                 try{msg.setContentObject(object);}
171                 catch (IOException e){e.printStackTrace();}}
172         sender.send(msg);
```

## Líneas de código utilizando el patrón *Implementation_JADE*

```
*lineas originales en un programa que usa el patron de implementacion
las demas lineas que no tienen * son las q harian falta para
realizar el trabajo de los agentes sin usar el patron de implementacion

// codigo implementado en el Server
*Init_Platform.init(true, null, false);
*Init_Platform.createAgentContainer("Server_Agent");
*Init_Platform.addAgent_ToContainer("Server_Agent", "Server_Agent", "test.client_server.Server_Agent", new Object[0]);
    *Work_DF.register_ByType_DF(this, "Tipo_Agente_Servidor");
    *addBehaviour(new ProcessMsg_Server(this, null));
        *System.out.println("Soy el Agente Servidor, el Agente Cliente dice: " + msg.getContent());
        *System.out.println("soy el Agente Servidor: mandando respuesta");
        *ACLMessage reply = msg.createReply();
        *reply.setContent("me alegro de que estes vivo");
        *myAgent.send(reply);

// codigo implementado en el Client
*Join_Platform.init("10.9.12.113");
*Join_Platform.createAgentContainer("Client_Agent", "1099");
*Join_Platform.addAgent_ToContainer("Client_Agent", "Client_Agent", "test.client_server.Client_Agent", new Object[0]);
    *addBehaviour(new ProcessMsg_Client(this, null));
        *System.out.println("Soy el Agente Cliente, el Agente Servidor se alegra de que yo este vivo");
    *System.out.println("Soy el Agente Cliente: mandando mensaje");
    *Work_ACL.sendMessage_ByType(this, "Tipo_Agente_Servidor", "ya estoy vivo");
```

# *Anexo 6: Prueba piloto*

Despliegue del Observatorio Tecnológico

Para ver el OT en funcionamiento con el SMA desplegado se desarrolló una prueba piloto con 38 usuarios. Las 38 personas escogidas son graduados de Ingeniería Informática y tienen experiencia en desarrollar búsquedas de información para enfocar sus investigaciones.

Los usuarios interactúan con la interfaz gráfica como se muestra en la Figura 1. Aquí el usuario especifica las palabras claves que quiere que aparezcan dentro de los documentos, además puede especificar las Uris que desea que sean revisadas. Esta información se recoge en el perfil del usuario, el cual puede ser actualizado y cambiado.

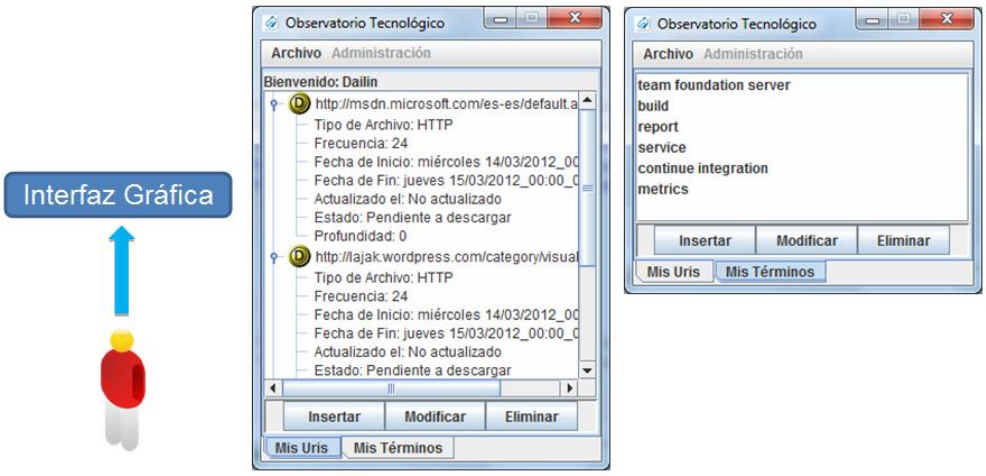



Figura 1. Interfaz gráfica donde el usuario hace su pedido de información

Los usuarios recibieron un correo electrónico con los resultados de la búsqueda, estos se obtienen de una iteración completa de búsqueda. El AP tiene un comportamiento proactivo y cada cierto tiempo revisa las fuentes de información de interés para el usuario, para ello el OT se apoya en el componente Observador. Cuando ocurren cambios en dichas fuentes de información relevantes a los intereses del usuario, el AP envía un nuevo correo electrónico con los hallazgos sin que medie una petición. También, si el usuario cambia sus intereses comienza

todo el proceso de búsqueda sin hacer una petición. De esta forma el OT explota su comportamiento proactivo para apoyar en la toma de decisiones.

En la prueba piloto se realizó una encuesta por cada correo electrónico enviado a los usuarios. Se tuvo en cuenta los 3 primeros correos. Las encuestas tenían como objetivo evaluar el funcionamiento del OT en varios aspectos: si los resultados responden a los intereses del usuario, que los resultados no se repitan, que los documentos se organicen de mayor relevancia a menos, regular el tamaño de los resultados obtenidos, periodicidad de la búsqueda, entre otros.

Las encuestas se diseñaron para que las afirmaciones fueran clasificadas como verdaderas (“V”) o falsas (“F”). En la Tabla 1 se muestra un resumen de los resultados de la encuesta aplicada a los 38 usuarios que participaron en la prueba piloto. En la Tabla 1 se muestran las preguntas y los resultados de las mismas en cada una de las iteraciones de la encuesta realizada. Los resultados se exponen con el porcentaje de las veces que tuvieron una evaluación “V”.

| Pregunta | 1ra encuesta | 2da encuesta | 3ra encuesta |
|---|---|---|---|
| Los documentos están relacionados con los intereses | 50 | 83.3 | 100 |
| Los documentos sirven para el trabajo en su proyecto | 41.6 | 50 | 71.4 |
| Los documentos están organizados de mayor a menor relevancia | 33.33 | 36.33 | 68.4 |
| El correo contiene documentos enviados anteriormente | 0 | 0 | 11,1 |
| El primer documento tiene unas alta relación con los intereses | 41.6 | 83.3 | 71.4 |
| Es adecuada a cantidad de documentos en el correo | 33.33 | 71.4 | 100 |
| Los metadatos resultaron de utilidad | --- | --- | 100 |

Tabla 1. Resumen del resultado de la encuesta.

Durante la prueba piloto y como resultado de las encuestas se realizaron actualizaciones en la implementación y ajustes en las configuraciones de los AP que a continuación se enumeran:

- Enviar un máximo de 50 documentos en cada correo.

- Los documentos que se envíen deben tener cada uno un valor numérico que represente la relevancia del documento.
- No enviar como adjunto el documento más relevante.
- Enviar un correo por cada semana.
- Enviar metadatos extras de interés para cada documento en el correo (autor, total de páginas, fecha de creación y modificación, palabras claves y un resumen del texto donde se resaltan las palabras buscadas).
- Cambiar el formato e implementación del envío del correo.
- Tener un fichero local por cada usuario para no reenviar documentos que fueron enviados antes.

# *Anexo 7: Grado de dificultad de las funciones utilizadas en los experimentos*

La Tabla 1 resume los grados de dificultad de estas funciones, mostrando la media aritmética (MA), la mediana (ME), el mínimo (MIN), el máximo (MAX) y la desviación estándar (DESV) de varias métricas medidas en cada función base. La métrica BFDC calcula el valor de la correlación entre la distancia en bits respecto al óptimo del bloque (1111) y la evaluación que recibe en las funciones de base. MAB mide el promedio de los valores asignados por la función a cada cantidad de bits correctos, mientras que la MABP es similar a MAB excepto en que es un promedio ponderado, teniendo en cuenta la cantidad de cadenas de 4 bits que tienen 0, 1, 2 o 3 bits iguales a 1111, que son 1, 4, 6 y 4 respectivamente. Las columnas 0-1, 0-2, 0-3, 1-2, 1-3 y 2-3, muestran la diferencia entre los valores asignados por la función a las cadenas con esas cantidades de bits. Por ejemplo, en F3210 (Deceptive) la diferencia entre el valor que le da la función a las soluciones con 1 bit correcto (2) respecto al valor para 3 bits correctos (0) es 2= 2-0. La tabla 3 muestra que las funciones escogidas tienen una variedad representativa, con distintos grados de dificultad. Por ejemplo, BFDC para F3210 (Deceptive) vale 1 y debe ser más difícil que F0123 (Onemax) en que BFDC vale -1.

Tabla 1: Métricas en las funciones base.

| | MA | ME | MIN | MAX | DESV |
|---|---|---|---|---|---|
| 0-1 | 0.75 | 0 | -1 | 3 | 1.43 |
| 0-2 | 0.39 | 0 | -3 | 3 | 1.75 |
| 0-3 | 0.21 | 0 | -3 | 3 | 1.71 |
| 1-2 | -0.36 | 0 | -2 | 1 | 0.73 |
| 1-3 | -0.54 | -1 | -3 | 2 | 1.37 |
| 2-3 | -0.18 | 0 | -3 | 2 | 1.39 |
| BFDC | -0.03 | 0 | -1 | 1 | 0.65 |
| MAB | 1.02 | 1 | 0 | 1.50 | 0.41 |
| MABP | 0.94 | 1 | 0 | 1.87 | 0.51 |

# *Anexo 8: Resultado de comparación de las metaheurísticas*

Media aritmética de los resultados de las 11 metaheurísticas en las 28 funciones, calculada sobre las 30 repeticiones

| MA | PGDA | PRRT | PTA | LS | EE5 | EE100 | AG100 | GDA | TA | RRT | PLS |
|---|---|---|---|---|---|---|---|---|---|---|---|
| f0000 | 99.87 | 100.00 | 100.00 | 100.00 | 98.93 | 85.07 | 86.93 | 79.73 | 100.00 | 77.33 | 98.93 |
| f0001 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 94.93 | 98.43 | 100.00 | 87.50 | 100.00 |
| f0003 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 97.40 | 99.13 | 99.60 | 87.90 | 86.73 | 100.00 |
| f0022 | 100.00 | 100.00 | 100.00 | 100.00 | 99.53 | 94.27 | 98.00 | 95.20 | 56.27 | 79.67 | 100.00 |
| f0111 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 99.80 | 91.20 | 88.03 | 100.00 | 80.20 | 99.70 |
| f0120 | 65.73 | 59.67 | 56.07 | 56.53 | 58.40 | 67.33 | 81.50 | 66.87 | 100.00 | 79.13 | 76.63 |
| f0122 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 97.27 | 95.00 | 58.97 | 76.07 | 99.93 |
| f0123 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 98.20 | 99.80 | 99.67 | 66.43 | 79.87 | 100.00 |
| f0131 | 91.80 | 96.00 | 100.00 | 77.40 | 84.63 | 87.07 | 87.07 | 81.33 | 100.00 | 85.27 | 85.33 |
| f1001 | 100.00 | 100.00 | 100.00 | 81.40 | 83.30 | 98.10 | 95.30 | 97.67 | 100.00 | 86.43 | 92.20 |
| f1002 | 100.00 | 100.00 | 100.00 | 70.90 | 84.50 | 99.93 | 97.93 | 63.00 | 65.87 | 65.43 | 94.53 |
| f1012 | 100.00 | 100.00 | 100.00 | 81.00 | 92.30 | 100.00 | 98.27 | 72.03 | 69.17 | 72.27 | 97.90 |
| f1111 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 99.50 | 90.20 | 85.60 | 100.00 | 84.00 | 99.20 |
| f1112 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 96.87 | 98.20 | 52.27 | 75.70 | 100.00 |
| f1120 | 66.07 | 60.33 | 56.40 | 56.33 | 57.87 | 66.67 | 81.37 | 66.30 | 100.00 | 78.07 | 77.87 |
| f1221 | 68.77 | 61.33 | 56.60 | 85.53 | 58.87 | 69.87 | 85.73 | 69.77 | 71.00 | 77.47 | 80.03 |
| f2000 | 98.13 | 98.93 | 100.00 | 75.93 | 85.67 | 83.33 | 84.80 | 86.07 | 100.00 | 75.93 | 84.47 |
| f2001 | 99.87 | 100.00 | 100.00 | 59.50 | 88.20 | 94.07 | 91.17 | 53.93 | 56.77 | 56.50 | 90.33 |
| f2012 | 98.27 | 97.87 | 98.60 | 92.27 | 92.27 | 97.80 | 97.00 | 97.53 | 51.00 | 79.17 | 93.87 |
| f2200 | 72.27 | 63.73 | 61.93 | 61.33 | 70.87 | 75.47 | 76.87 | 74.47 | 100.00 | 82.93 | 72.13 |
| f2220 | 61.80 | 60.20 | 56.60 | 56.27 | 83.73 | 79.27 | 77.87 | 63.07 | 100.00 | 81.60 | 75.33 |
| f3000 | 95.87 | 98.33 | 99.97 | 87.23 | 88.40 | 89.60 | 86.27 | 88.63 | 87.87 | 88.13 | 90.70 |
| f3001 | 95.73 | 95.90 | 97.20 | 93.53 | 93.57 | 92.63 | 90.87 | 93.77 | 87.50 | 87.13 | 94.67 |
| f3002 | 94.23 | 94.53 | 94.03 | 78.90 | 93.43 | 93.77 | 92.80 | 70.27 | 72.10 | 73.37 | 94.20 |
| f3010 | 96.53 | 97.27 | 99.27 | 59.37 | 95.93 | 95.30 | 90.07 | 55.07 | 56.53 | 56.87 | 95.93 |
| f3012 | 93.93 | 95.37 | 96.03 | 90.43 | 95.40 | 94.80 | 94.87 | 88.57 | 72.40 | 82.27 | 95.50 |
| f3102 | 95.20 | 94.43 | 93.63 | 88.03 | 91.43 | 92.17 | 91.23 | 95.10 | 51.20 | 83.00 | 87.53 |
| f3210 | 85.13 | 81.00 | 98.57 | 78.03 | 83.17 | 83.40 | 83.57 | 83.17 | 100.00 | 80.73 | 82.47 |

Mediana de los resultados de las 11 metaheurísticas en las 28 funciones, calculada sobre las 30 repeticiones

| MA | PGDA | PRRT | PTA | LS | EE5 | EE100 | AG100 | GDA | TA | RRT | PLS |
|---|---|---|---|---|---|---|---|---|---|---|---|
| f0000 | 100 | 100 | 100 | 100 | 100 | 84 | 88 | 84 | 100 | 76 | 100 |
| f0001 | 100 | 100 | 100 | 100 | 100 | 100 | 96 | 100 | 100 | 87 | 100 |
| f0003 | 100 | 100 | 100 | 100 | 100 | 97.5 | 99 | 100 | 88 | 86 | 100 |
| f0022 | 100 | 100 | 100 | 100 | 100 | 94 | 98 | 96 | 56 | 79 | 100 |
| f0111 | 100 | 100 | 100 | 100 | 100 | 100 | 91 | 91 | 100 | 79 | 100 |
| f0120 | 65 | 58 | 56 | 56 | 58 | 68 | 82 | 67 | 100 | 79 | 76 |
| f0122 | 100 | 100 | 100 | 100 | 100 | 100 | 98 | 96 | 59 | 76 | 100 |
| f0123 | 100 | 100 | 100 | 100 | 100 | 98 | 100 | 100 | 66.5 | 80 | 100 |
| f0131 | 92 | 97 | 100 | 77 | 85 | 86.5 | 87.5 | 81 | 100 | 85.5 | 84 |
| f1001 | 100 | 100 | 100 | 82 | 83.5 | 100 | 96.5 | 100 | 100 | 87 | 94 |
| f1002 | 100 | 100 | 100 | 71.5 | 85 | 100 | 98 | 63.5 | 66 | 65 | 95.5 |
| f1012 | 100 | 100 | 100 | 81 | 91 | 100 | 98 | 71.5 | 69 | 72 | 98.5 |
| f1111 | 100 | 100 | 100 | 100 | 100 | 100 | 91 | 88 | 100 | 85 | 100 |
| f1112 | 100 | 100 | 100 | 100 | 100 | 100 | 98 | 100 | 52 | 76 | 100 |
| f1120 | 66 | 60 | 56 | 56 | 58 | 67 | 82 | 67.5 | 100 | 78 | 78.5 |
| f1221 | 69 | 60 | 56 | 85 | 58 | 70 | 86 | 68.5 | 71 | 77.5 | 80 |
| f2000 | 98 | 100 | 100 | 76 | 86 | 84 | 85 | 86 | 100 | 76 | 85 |
| f2001 | 100 | 100 | 100 | 60 | 88 | 94 | 91.5 | 54 | 56 | 56.5 | 90 |
| f2012 | 98 | 98 | 98 | 92 | 92 | 98 | 98 | 98 | 51 | 79 | 94 |
| f2200 | 72 | 63 | 62 | 60 | 72 | 75 | 76 | 74 | 100 | 83 | 72 |
| f2220 | 62 | 60 | 57 | 56 | 84 | 80 | 78 | 63 | 100 | 81 | 75 |
| f3000 | 96 | 99 | 100 | 87 | 89 | 89.5 | 86 | 89.5 | 88 | 89 | 91 |
| f3001 | 96 | 96 | 97 | 94 | 93 | 93 | 91 | 93 | 87.5 | 87 | 95 |
| f3002 | 94.5 | 94 | 93.5 | 79 | 93.5 | 93.5 | 93 | 69.5 | 72 | 73.5 | 94.5 |
| f3010 | 96 | 98 | 99 | 58.5 | 96 | 96 | 91 | 54 | 56 | 56 | 96 |
| f3012 | 95 | 95 | 96.5 | 90 | 95 | 94 | 95.5 | 88 | 72 | 82 | 96 |
| f3102 | 95.5 | 94.5 | 93.5 | 88.5 | 92 | 92 | 92 | 95.5 | 51 | 84 | 87.5 |
| f3210 | 85 | 81 | 99 | 78 | 83 | 84 | 84.5 | 84 | 100 | 81 | 83 |

Desviación estándar de los resultados de las 11 metaheurísticas en las 28 funciones, calculada sobre las 30 repeticiones

| MA | PGDA | PRRT | PTA | LS | EE5 | EE100 | AG100 | GDA | TA | RRT | PLS |
|---|---|---|---|---|---|---|---|---|---|---|---|
| f0000 | 0.73 | 0.00 | 0.00 | 0.00 | 2.08 | 3.47 | 7.42 | 12.78 | 0.00 | 6.50 | 2.56 |
| f0001 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 3.55 | 2.53 | 0.00 | 3.26 | 0.00 |
| f0003 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 1.10 | 1.07 | 0.62 | 0.80 | 1.53 | 0.00 |
| f0022 | 0.00 | 0.00 | 0.00 | 0.00 | 0.86 | 1.46 | 1.58 | 3.43 | 1.95 | 3.02 | 0.00 |
| f0111 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.76 | 4.16 | 9.73 | 0.00 | 5.12 | 1.21 |
| f0120 | 5.06 | 3.57 | 3.84 | 3.79 | 3.65 | 3.94 | 3.74 | 6.03 | 0.00 | 3.62 | 7.02 |
| f0122 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 2.32 | 4.32 | 1.69 | 2.43 | 0.37 |
| f0123 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.76 | 0.41 | 0.48 | 1.77 | 1.93 | 0.00 |
| f0131 | 3.06 | 3.87 | 0.00 | 1.54 | 1.92 | 2.02 | 2.72 | 1.95 | 0.00 | 1.80 | 5.14 |
| f1001 | 0.00 | 0.00 | 0.00 | 7.19 | 5.44 | 3.00 | 3.56 | 3.63 | 0.00 | 3.56 | 3.91 |
| f1002 | 0.00 | 0.00 | 0.00 | 3.12 | 4.60 | 0.37 | 2.23 | 4.98 | 2.30 | 3.07 | 4.73 |
| f1012 | 0.00 | 0.00 | 0.00 | 1.97 | 4.22 | 0.00 | 1.89 | 3.44 | 3.01 | 2.79 | 2.38 |
| f1111 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 1.14 | 6.45 | 10.43 | 0.00 | 3.18 | 1.92 |
| f1112 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 3.30 | 2.27 | 2.10 | 2.73 | 0.00 |
| f1120 | 5.98 | 4.00 | 3.34 | 4.43 | 3.52 | 2.84 | 4.37 | 5.63 | 0.00 | 3.69 | 6.93 |
| f1221 | 6.16 | 3.87 | 3.29 | 5.40 | 3.18 | 4.36 | 4.23 | 6.24 | 1.98 | 1.76 | 4.92 |
| f2000 | 2.46 | 2.15 | 0.00 | 4.47 | 5.41 | 3.38 | 5.40 | 5.74 | 0.00 | 4.47 | 6.43 |
| f2001 | 0.51 | 0.00 | 0.00 | 3.15 | 4.21 | 3.26 | 3.74 | 3.76 | 3.13 | 3.77 | 4.67 |
| f2012 | 2.02 | 1.74 | 1.07 | 3.18 | 3.70 | 2.06 | 2.67 | 2.27 | 2.41 | 2.88 | 3.71 |
| f2200 | 5.48 | 4.92 | 3.54 | 3.94 | 4.29 | 4.93 | 3.59 | 4.29 | 0.00 | 3.31 | 6.10 |
| f2220 | 3.34 | 4.08 | 3.02 | 3.35 | 2.96 | 3.26 | 5.30 | 3.96 | 0.00 | 4.21 | 5.21 |
| f3000 | 2.29 | 1.95 | 0.18 | 2.37 | 2.50 | 2.70 | 4.16 | 4.51 | 2.34 | 2.94 | 3.43 |
| f3001 | 1.28 | 1.71 | 1.00 | 2.56 | 2.21 | 2.17 | 3.93 | 2.28 | 2.42 | 3.36 | 2.41 |
| f3002 | 2.11 | 2.53 | 1.90 | 2.78 | 2.11 | 2.28 | 3.55 | 3.45 | 2.94 | 2.28 | 2.33 |
| f3010 | 1.68 | 1.68 | 0.78 | 2.99 | 1.48 | 1.88 | 3.15 | 2.56 | 2.18 | 2.84 | 2.52 |
| f3012 | 2.20 | 1.73 | 1.73 | 2.06 | 2.04 | 1.88 | 2.89 | 3.01 | 3.23 | 2.18 | 2.01 |
| f3102 | 1.77 | 2.71 | 1.79 | 2.61 | 2.79 | 2.32 | 4.22 | 1.81 | 2.68 | 2.85 | 2.56 |
| f3210 | 2.43 | 1.97 | 1.19 | 1.81 | 2.53 | 2.27 | 3.44 | 2.29 | 0.00 | 3.52 | 2.64 |

Suma de resultados de las comparaciones dos a dos de las 11 metaheurísticas en las 28 funciones, usando la prueba no pareada de Wilcoxon o Prueba de Mann-Whitney con significación de 0.05 calculada sobre las 30 repeticiones

| MA | PGDA | PRRT | PTA | LS | EE5 | EE100 | AG100 | GDA | TA | RRT | PLS |
|---|---|---|---|---|---|---|---|---|---|---|---|
| f0000 | 4 | 4 | 4 | 4 | 4 | -6 | -5 | -8 | 4 | -9 | 4 |
| f0001 | 3 | 3 | 3 | 3 | 3 | 3 | -8 | -6 | 3 | -10 | 3 |
| f0003 | 5 | 5 | 5 | 5 | 5 | -6 | -3 | -3 | -8 | -10 | 5 |
| f0022 | 5 | 5 | 5 | 5 | 5 | -5 | -2 | -5 | -10 | -8 | 5 |
| f0111 | 3 | 3 | 3 | 3 | 3 | 3 | -7 | -7 | 3 | -10 | 3 |
| f0120 | 0 | -5 | -9 | -8 | -6 | 0 | 8 | 0 | 10 | 6 | 4 |
| f0122 | 4 | 4 | 4 | 4 | 4 | 4 | -4 | -6 | -10 | -8 | 4 |
| f0123 | 4 | 4 | 4 | 4 | 4 | -6 | 3 | -3 | -10 | -8 | 4 |
| f0131 | 4 | 6 | 9 | -10 | -4 | 1 | 1 | -8 | 9 | -4 | -4 |
| f1001 | 7 | 7 | 7 | -9 | -9 | 1 | -2 | 1 | 7 | -6 | -4 |
| f1002 | 7 | 7 | 7 | -4 | -2 | 7 | 2 | -10 | -7 | -7 | 0 |
| f1012 | 7 | 7 | 7 | -4 | -2 | 7 | 1 | -7 | -10 | -7 | 1 |
| f1111 | 3 | 3 | 3 | 3 | 3 | 3 | -7 | -7 | 3 | -10 | 3 |
| f1112 | 4 | 4 | 4 | 4 | 4 | 4 | -5 | -5 | -10 | -8 | 4 |
| f1120 | 0 | -4 | -8 | -8 | -8 | 0 | 7 | 0 | 10 | 5 | 6 |
| f1221 | -1 | -6 | -10 | 9 | -8 | -1 | 9 | -1 | -1 | 4 | 6 |
| f2000 | 5 | 5 | 9 | -9 | -1 | -4 | -2 | -1 | 9 | -9 | -2 |
| f2001 | 8 | 8 | 8 | -4 | -1 | 4 | 1 | -10 | -7 | -7 | 0 |
| f2012 | 5 | 5 | 6 | -4 | -4 | 5 | 4 | 5 | -10 | -8 | -4 |
| f2200 | 0 | -8 | -8 | -8 | -2 | 2 | 5 | 1 | 10 | 8 | 0 |
| f2220 | -4 | -5 | -9 | -9 | 8 | 3 | 2 | -3 | 10 | 6 | 1 |
| f3000 | 6 | 8 | 10 | -6 | -3 | 0 | -7 | -1 | -5 | -4 | 2 |
| f3001 | 6 | 7 | 10 | 0 | 0 | -2 | -5 | 0 | -9 | -9 | 2 |
| f3002 | 4 | 4 | 4 | -4 | 4 | 4 | 4 | -10 | -7 | -7 | 4 |
| f3010 | 4 | 8 | 10 | -4 | 3 | 2 | -2 | -10 | -7 | -7 | 3 |
| f3012 | 0 | 5 | 6 | -4 | 5 | 3 | 4 | -6 | -10 | -8 | 5 |
| f3102 | 8 | 7 | 5 | -5 | 0 | 0 | 0 | 8 | -10 | -8 | -5 |
| f3210 | 5 | -7 | 8 | -10 | 0 | 0 | 1 | 0 | 10 | -7 | 0 |

Porciento de convergencia al óptimo de las 11 metaheurísticas en las 28 funciones, en las 30 repeticiones.

| MA | PGDA | PRRT | PTA | LS | EE5 | EE100 | AG100 | GDA | TA | RRT | PLS |
|---|---|---|---|---|---|---|---|---|---|---|---|
| f0000 | 97% | 100% | 100% | 100% | 77% | 0% | 3% | 0% | 100% | 0% | 80% |
| f0001 | 100% | 100% | 100% | 100% | 100% | 100% | 13% | 63% | 100% | 0% | 100% |
| f0003 | 100% | 100% | 100% | 100% | 100% | 0% | 43% | 67% | 0% | 0% | 100% |
| f0022 | 100% | 100% | 100% | 100% | 77% | 0% | 27% | 10% | 0% | 0% | 100% |
| f0111 | 100% | 100% | 100% | 100% | 100% | 93% | 0% | 0% | 100% | 0% | 93% |
| f0120 | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 100% | 0% | 0% |
| f0122 | 100% | 100% | 100% | 100% | 100% | 100% | 27% | 13% | 0% | 0% | 97% |
| f0123 | 100% | 100% | 100% | 100% | 100% | 3% | 80% | 67% | 0% | 0% | 100% |
| f0131 | 0% | 23% | 100% | 0% | 0% | 0% | 0% | 0% | 100% | 0% | 0% |
| f1001 | 100% | 100% | 100% | 0% | 0% | 63% | 20% | 63% | 100% | 0% | 7% |
| f1002 | 100% | 100% | 100% | 0% | 0% | 97% | 37% | 0% | 0% | 0% | 17% |
| f1012 | 100% | 100% | 100% | 0% | 3% | 100% | 43% | 0% | 0% | 0% | 50% |
| f1111 | 100% | 100% | 100% | 100% | 100% | 83% | 3% | 3% | 100% | 0% | 83% |
| f1112 | 100% | 100% | 100% | 100% | 100% | 100% | 27% | 53% | 0% | 0% | 100% |
| f1120 | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 100% | 0% | 0% |
| f1221 | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% |
| f2000 | 40% | 70% | 100% | 0% | 3% | 0% | 0% | 0% | 100% | 0% | 0% |
| f2001 | 93% | 100% | 100% | 0% | 0% | 3% | 0% | 0% | 0% | 0% | 0% |
| f2012 | 43% | 27% | 33% | 3% | 7% | 27% | 23% | 23% | 0% | 0% | 7% |
| f2200 | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 100% | 0% | 0% |
| f2220 | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 100% | 0% | 0% |
| f3000 | 7% | 40% | 97% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% |
| f3001 | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% |
| f3002 | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% |
| f3010 | 7% | 0% | 47% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 7% |
| f3012 | 0% | 3% | 3% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% |
| f3102 | 0% | 3% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% |
| f3210 | 0% | 0% | 27% | 0% | 0% | 0% | 0% | 0% | 100% | 0% | 0% |

Rangos usados para prueba de Friedman y de Wilcoxon pareada, sobre la base de 10 métricas de comparación.

| MA | PC | MA(MA) | ME(MA) | MA(DT) | MA(SW) | ME(SW) | SW | MA(PC) | ME(PC) | SR Friedman en comparación de medias |
|---|---|---|---|---|---|---|---|---|---|---|
| PGDA | 4 | 1 | 3 | 4 | 1 | 3 | 1 | 3 | 3 | 2 |
| PRRT | 2 | 4 | 2 | 3 | 3 | 2 | 1 | 2 | 2 | 3 |
| PTA | 1 | 2 | 1 | 1 | 2 | 1 | 3 | 1 | 1 | 1 |
| LS | 5 | 8 | 9 | 6 | 9 | 8 | 10 | 6 | 5 | 9 |
| EE5 | 6 | 7 | 6 | 7 | 6 | 7 | 7 | 7 | 5 | 6 |
| EE100 | 8 | 6 | 5 | 5 | 5 | 5 | 3 | 8 | 5 | 5 |
| AG100 | 9 | 5 | 7 | 10 | 7 | 6 | 3 | 10 | 5 | 7 |
| GDA | 9 | 9 | 10 | 11 | 10 | 8 | 9 | 9 | 5 | 10 |
| TA | 3 | 10 | 8 | 2 | 8 | 10 | 7 | 4 | 5 | 8 |
| RRT | 9 | 11 | 11 | 9 | 11 | 11 | 10 | 11 | 5 | 11 |
| PLS | 7 | 3 | 4 | 8 | 4 | 4 | 3 | 5 | 4 | 4 |

| | |
|---|---|
| Valor de S | 76.086 |
| Grados de libertad | 10 |
| p-value | $2.92^{*}\ 10^{-12}$ |